Bölüm 7: Test ve Hata Ayıklama

Java ile Nesne Yönelimli Programlama

Test türleri · Birim testi ve JUnit · Doğrulama metotları · Sınır değerleri · Regresyon testi · Otomatik test · Hata ayıklayıcı · Profil oluşturucu · Kod kapsamı · Statik analiz · Özellik tabanlı test · Bellek analizi

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Test türleri

Birim, modül, alt sistem, sistem ve kabul testini ayırt etmek; her birinin neyi doğruladığını açıklamak.

JUnit ile birim testi

`@Test`, `@BeforeEach` ve `assertEquals`, `assertThrows` gibi doğrulama metotlarıyla otomatik testler yazmak, başarısız testin mesajını okumak.

İyi test tasarımı

Sınır değerlerini seçmek, parametreli test yazmak; testleri belirleyici ve tekrarlanabilir kurmak; regresyon testinin değerini açıklamak.

Hata ayıklama

Kesme noktası koymak, programı adım adım (Step Into / Over / Out) yürütmek, değişkenleri izleyerek hatanın kaynağını bulmak.

Analiz araçları

Profil oluşturucu, kod kapsamı, statik analiz (SpotBugs), özellik tabanlı test, Selenium, Memory Analyzer ve VisualVM'in hangi soruyu yanıtladığını bilmek.

Doğru araç, doğru soru

Bir sorun için test mi, hata ayıklayıcı mı, profil oluşturucu mu gerektiğine karar vermek.

GİRİŞ

Test ve Hata Ayıklama: Neden?

"Test, hata bulmakla ilgili değildir; sistemin beklendiği gibi çalıştığından emin olmakla ilgilidir."

Michael Bolton (yazılım test uzmanı)

"Geleceği tahmin etmenin en iyi yolu onu yaratmaktır."

Sıklıkla Abraham Lincoln'e atfedilir; Peter Drucker ve Alan Kay'e de dayandırılır

"Hata (bug), henüz keşfedilmemiş bir özelliktir."

Anonim yazılımcı esprisi

"Kalite bir eylem değil, bir alışkanlıktır."

Aristoteles'e atfedilir; aslında Will Durant'ın Aristoteles yorumudur

"Test etmemek, yazılımla Rus ruleti oynamaya benzer."

Anonim

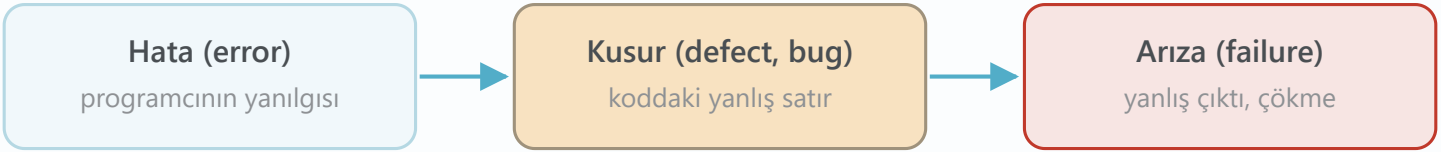
"Test, hataların varlığını gösterebilir; yokluğunu asla gösteremez."

Edsger W. Dijkstra

Test, programın **beklenen davranışı** gösterdiğini sistematik olarak sınamaktır. Ama hiçbir test kümesi hatasızlığı **kanıtlamaz**; yalnızca güveni artırır.

GİRİŞ

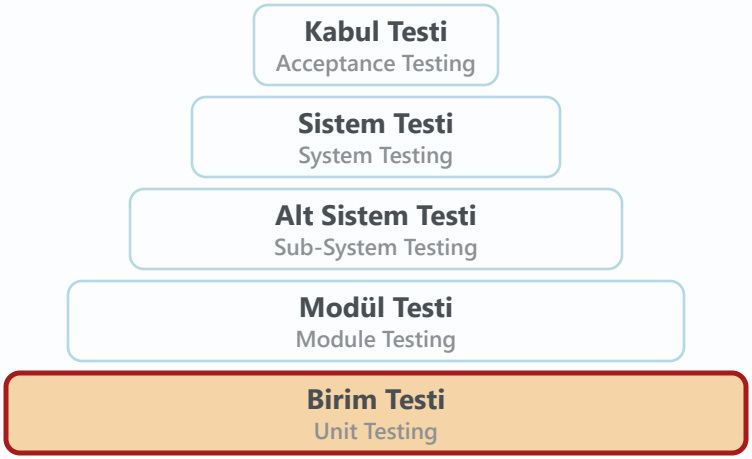
Hata, Kusur, Arıza: Test ile Hata Ayıklama Farkı



Test arızayı ortaya çıkarır → hata ayıklama kusuru bulur → düzeltme hatayı giderir

	Test (testing)	**Hata ayıklama (debugging)**
Amaç	Programın beklendiği gibi çalışıp çalışmadığını **sınamak**	Bilinen bir arızanın **nedenini bulup** düzeltmek
Soru	"Çalışıyor mu?"	"Neden çalışmıyor?"
Sonuç	Geçti / kaldı (✓ / ✗)	Hatalı satır ve düzeltme
Araçlar	JUnit, Selenium, kod kapsamı	Hata ayıklayıcı (debugger), yığın izi, `println`
Ne zaman	Sürekli, her değişiklikten sonra	Bir test kaldığında ya da kullanıcı hata bildirdiğinde

Yazılım Geliştirmede Test Türleri



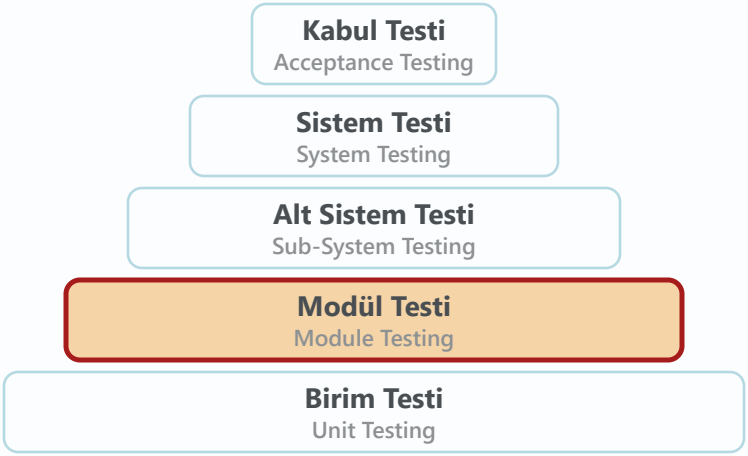
aşağıdan yukarı: küçük parça → bütün sistem

Birim Testi (Unit Testing)

- **Bireysel, bağımsız bileşenlerin** (bir sınıf, bir metot) test edilmesine odaklanır.
- Her bileşenin **tek başına** doğru çalıştığından emin olunur.
- Örnek: `BankaHesabi.cek()` yetersiz bakiyede istisna fırlatıyor mu?

Birim testi: en küçük parça, en çok sayıda test, en hızlı çalışan.

Yazılım Geliştirmede Test Türleri



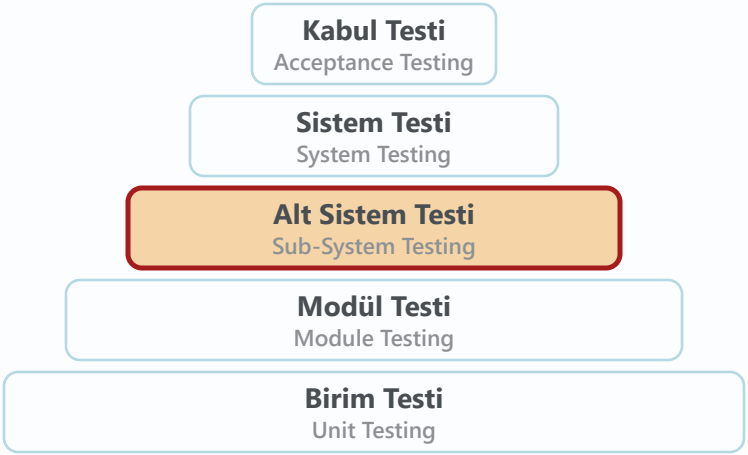
aşağıdan yukarı: küçük parça → bütün sistem

Modül Testi (Module Testing)

- Birbiriyle ilgili bileşenlerden oluşan bir **modülün** test edilmesidir.
- Modül içindeki bileşenlerin **entegrasyonu** ve işlevselliği doğrulanır.
- Örnek: `Hesap`, `Islem` ve `Dekont` sınıfları birlikte para transferini doğru yapıyor mu?

Modül testi: ilgili bileşenler bir arada.

Yazılım Geliştirmede Test Türleri



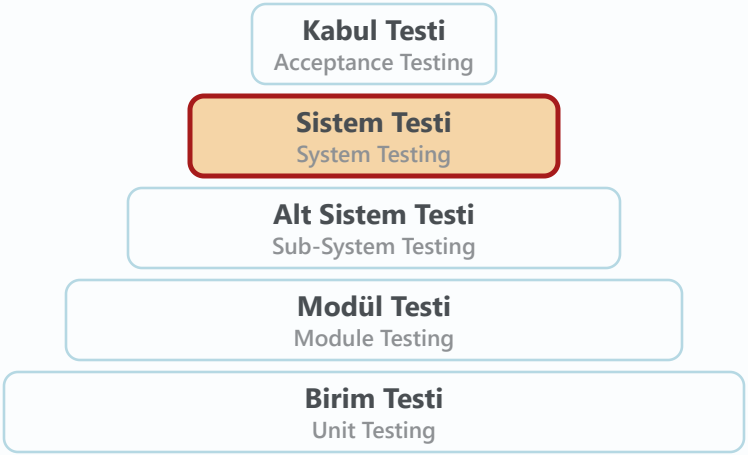
aşağıdan yukarı: küçük parça → bütün sistem

Alt Sistem Testi (Sub-System Testing)

- Alt sistemler arasındaki **arayüz uyumsuzluklarını** test eder.
- Bileşenler arasındaki **iletişim ve veri akışının** doğruluğunu kontrol eder.
- Örnek: hesap alt sistemi ile veri tabanı alt sistemi aynı tarih biçimini mi kullanıyor?

Alt sistem testi: alt sistemlerin birleştiği arayüzler.

Yazılım Geliştirmede Test Türleri



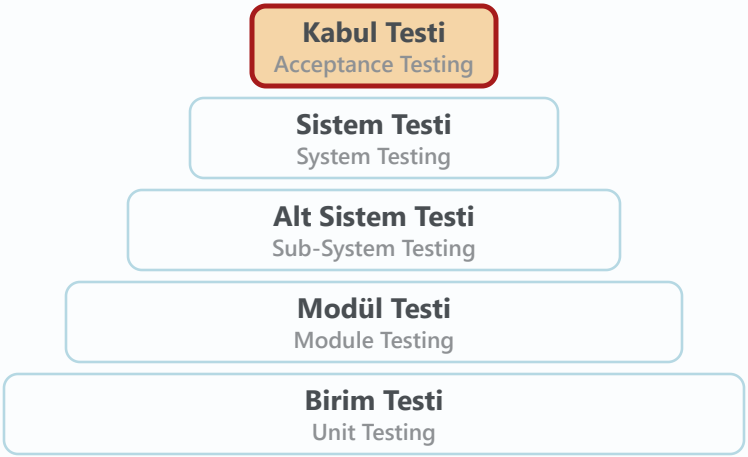
aşağıdan yukarı: küçük parça → bütün sistem

Sistem Testi (System Testing)

- Alt sistemler arasındaki **etkileşimlerin** bütün olarak test edilmesidir.
- **İşlevsel** (ne yapıyor) ve **işlevsel olmayan** (hız, güvenlik, kullanılabilirlik) gereksinimlerin karşılandığı doğrulanır.
- Örnek: 1000 eş zamanlı kullanıcıyla havale 2 saniyenin altında tamamlanıyor mu?

Sistem testi: bütün sistem, işlevsel ve işlevsel olmayan gereksinimler.

Yazılım Geliştirmede Test Türleri



aşağıdan yukarı: küçük parça → bütün sistem

Kabul Testi (Acceptance Testing)

- Simüle veriler yerine **gerçek verilerle** sistemin test edilmesidir.
- Sistemin gerçek dünyadaki **kullanılabilirliğini ve performansını** doğrular.
- **Kullanıcılar ile beraber** yapılır: **alfa** testi geliştiricinin ortamında, **beta** testi kullanıcının kendi ortamında.

Kabul testi: müşteri “bu sistemi kabul ediyorum” diyebiliyor mu?

TEST TÜRLERİ

Test Düzeyleri Bir Bakışta

Düzey	Neyi test eder?	Kim yapar?	Veri
Birim	Tek sınıf / metot	Geliştirici	Küçük, elle seçilmiş
Modül	Birlikte çalışan birkaç sınıf	Geliştirici	Senaryo verisi
Alt sistem	Alt sistemler arası arayüzler	Geliştirici, test ekibi	Senaryo verisi
Sistem	Bütün sistem; hız, güvenlik	Test ekibi	Gerçeğe yakın
Kabul (alfa / beta)	Müşteri gereksinimleri	Kullanıcı ile birlikte	Gerçek veri

Kara kutu (black-box) testi

Kodun içine bakmadan, yalnızca **girdi → beklenen çıktı** ile sınanır. Kabul ve sistem testleri çoğunlukla böyledir.

Beyaz kutu (white-box) testi

Kodun yapısı bilinerek her dal ve döngü sınanır. Birim testleri ve kod kapsamı bu bakış açısını kullanır.

Aşağı düzeydeki hata ucuz ve hızlı bulunur; aynı hata kabul testinde ya da kullanıcıda ortaya çıkarsa düzeltmesi çok daha pahalıdır.

Önce: Elle Test Etmek

TEST EDİLECEK SINIF

```
public class HesapMakinesi {
  public int topla(int a, int b) {
    return a + b;
  }

  public int carp(int a, int b) {
    return a * b;
  }
}
```

ELLE DENEME PROGRAMI

```
public class ElleTest {
  public static void main(String[] args) {
    HesapMakinesi hm = new HesapMakinesi();
    System.out.println("topla(2, 3) = " + hm.topla(2, 3));
    System.out.println("carp(2, -3) = " + hm.carp(2, -3));
    // Doğru mu? Ekrana bakıp kendimiz karar veriyoruz.
  }
}
```

Konsol

```
topla(2, 3) = 5
carp(2, -3) = -6
```

BU YÖNTEMİN SORUNLARI

- Sonucun doğru olduğuna **insan gözü** karar verir; yorulur, yanılır.
- Kod değişince her şeyi **yeniden elle** denemek gerekir.
- Test kodu `main` içinde kalır, sonra silinir: emek kaybolur.
- 100 metot için 100 çıktı satırını kim okuyacak?

Çözüm: beklenen değeri **kodun içine yazıp** karşılaştırmayı bilgisayara bırakmak: **otomatik birim testi**.

JUnit

- **JUnit**, Java için en yaygın **birim testi çerçevesidir** (framework). Testler sıradan Java metotları olarak yazılır, JUnit onları bulur, çalıştırır ve raporlar.
- Bir **birim**, Java'da genellikle bir **sınıf** anlamına gelir; testler o sınıfın metotlarını tek tek sınar.
- JUnit temel olarak birim testi için tasarlanmıştır ama oldukça **esnektir**; şu testlerde de kullanılabilir:
  - **Modül testi**: birden fazla sınıfı içeren senaryoların test edilmesi.
  - **Sistem testi**: sistem düzeyindeki etkileşimlerin test edilmesi.
- IntelliJ IDEA, Eclipse, NetBeans, VS Code ve Maven / Gradle JUnit testlerini doğrudan çalıştırır.

Sürüm	Test nasıl yazılır?
JUnit 3	`TestCase` sınıfından kalıtım, adı `test` ile başlayan metotlar
JUnit 4	`@Test` etiketi, `org.junit`
JUnit 5	`@Test`, `org.junit.jupiter.api`; lambda, parametreli test

Örneklerde **JUnit 5** (Jupiter) kullanılır. Gradle ile projeye ekleme:

```
testImplementation("org.junit.jupiter:junit-jupiter:5.11.4")
```

Test sınıfları `src/test/java` klasörüne, test edilen sınıflar `src/main/java` klasörüne konur.

İlk JUnit 5 Testi

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

class HesapMakinesiTest {

  @Test
  void ikiSayiyiToplar() {
    HesapMakinesi hm = new HesapMakinesi();
    assertEquals(5, hm.topla(2, 3));
  }

  @Test
  void negatifSayiylaCarpar() {
    HesapMakinesi hm = new HesapMakinesi();
    assertEquals(-6, hm.carp(2, -3));
  }
}
```

Test Sonuçları

✓ `ikiSayiyiToplar()`
✓ `negatifSayiylaCarpar()`

2 test · 2 başarılı · 0 başarısız

`@Test`

Bu metodun bir test olduğunu JUnit'e söyler. Metot `void` ve parametresizdir; `main` gerekmez; `public` olması da gerekmez.

`assertEquals(beklenen, gerçek)`

Önce **beklenen** değer, sonra **gerçek** sonuç. Eşit değilse test **kalır** (✗). `import static` sayesinde sınıf adı yazılmaz.

Test adı = davranış

`ikiSayiyiToplar` gibi adlar, test kaldığında neyin bozulduğunu söyler.

Doğrulama (Assertion) Metotları

Metot	Ne zaman geçer?	Örnek (hepsi geçer)
`assertEquals(b, g)`	`b` ve `g` eşit (`equals`)	`assertEquals(5, 2 + 3)`
`assertEquals(b, g, delta)`	Ondalık sayılarda fark ≤ delta	`assertEquals(0.3, 0.1 + 0.2, 1e-9)`
`assertNotEquals(a, g)`	Eşit değil	`assertNotEquals(4, 2 + 3)`
`assertTrue` / `assertFalse`	Koşul doğru / yanlış	`assertFalse("abc".isEmpty())`
`assertNull` / `assertNotNull`	Değer `null` / `null` değil	`assertNotNull("x")`
`assertSame(a, b)`	Aynı nesne (`==`)	`assertSame(s, s)`
`assertArrayEquals`	Diziler öğe öğe eşit	`assertArrayEquals(new int[] {1, 2}, new int[] {1, 2})`
`assertThrows(T.class, kod)`	Kod o tipte istisna fırlatır	`assertThrows(ArithmeticException.class, () -> …)`
`assertAll(…)`	Tüm doğrulamalar (hepsi denenir)	`assertAll(() -> …, () -> …)`
`fail("mesaj")`	Asla; testi bilerek düşürür	`fail("henüz yazılmadı")`

Ondalıkta delta unutulursa `assertEquals(0.3, 0.1 + 0.2)` kalır: `expected: <0.3> but was: <0.30000000000000004>`. Son parametre isteğe bağlı bir mesajdır: `assertEquals(4, 2 + 3, "toplam yanlış")` → `toplam yanlış ==> expected: <4> but was: <5>`

Test Edilecek Sınıf: `BankaHesabi`

```java
public class BankaHesabi {
  private double bakiye;
  public BankaHesabi(double bakiye) {
    this.bakiye = bakiye;
  }
  public void yatir(double miktar) {
    if (miktar <= 0) {
      throw new IllegalArgumentException("Geçersiz miktar");
    }
    bakiye += miktar;
  }
  public void cek(double miktar) {
    if (miktar > bakiye) {
      throw new IllegalStateException("Yetersiz bakiye");
    }
    bakiye -= miktar;
  }
  public double getBakiye() {
    return bakiye;
  }
}
```

HANGİ DAVRANIŞLAR TEST EDİLMELİ?

Çağrı (bakiye 100)	Beklenen
`yatir(50)`	bakiye 150
`cek(30)`	bakiye 70
`cek(500)`	`IllegalStateException`; bakiye 100 kalır
`yatir(-10)`	`IllegalArgumentException`
`yatir(0)` (sınır)	`IllegalArgumentException`
`cek(100)` (sınır)	bakiye 0, istisna yok

Her satır ayrı bir test metodu olur. Hem **normal** durumlar hem **hatalı girdiler** hem de **sınırlar** düşünülür.

Bir Testin Yapısı: Hazırla, Çalıştır, Doğrula

```java
class AaaTest {

  @Test
  void cekmeBakiyeyiAzaltir() {
    // Hazırla (Arrange)
    BankaHesabi hesap = new BankaHesabi(100);
    // Çalıştır (Act)
    hesap.cek(30);
    // Doğrula (Assert)
    assertEquals(70.0, hesap.getBakiye(), 0.001);
  }
}
```

Test Sonuçları

✓ `cekmeBakiyeyiAzaltir()`

1 test · 1 başarılı

1 · Hazırla

Arrange: nesneyi ve verileri kur.

2 · Çalıştır

Act: test edilen metodu **bir kez** çağır.

3 · Doğrula

Assert: sonucu beklenenle karşılaştır.

Her test **tek bir davranışı** sınar. Bakiye `double` olduğu için `delta = 0.001` verilir.

Ortak Hazırlık: `@BeforeEach`

`BankaHesabiTest`

```
class BankaHesabiTest {
  private BankaHesabi hesap;

  @BeforeEach
  void hazirla() {
    hesap = new BankaHesabi(100);
  }

  @Test
  void yatirmaBakiyeyiArtirir() {
    hesap.yatir(50);
    assertEquals(150.0, hesap.getBakiye(), 0.001);
  }

  @Test
  void fazlaCekmeIstisnaFirlatir() {
    assertThrows(IllegalStateException.class,
      () -> hesap.cek(500));
    assertEquals(100.0, hesap.getBakiye(), 0.001);
  }
}
```

- `@BeforeEach` metodu **her testten önce** çalışır: her test **yeni ve temiz** bir hesapla (bakiye 100) başlar.
- JUnit **her test metodu için test sınıfından yeni bir nesne** oluşturur; bir testin değiştirdiği `hesap` diğerini etkilemez.
- `assertThrows`, ikinci parametredeki kodu çalıştırır ve o tipte bir istisna fırlatılmasını **bekler**. `() -> hesap.cek(500)` bir **lambda ifadesidir**: "çalıştırılacak kod parçası".
- İstisnadan sonra bakiyenin **değişmediği** de doğrulanır.

Test Sonuçları

✓ `fazlaCekmeIstisnaFirlatir()`
✓ `yatirmaBakiyeyiArtirir()`

2 test · 2 başarılı

JUnit Testleri Nasıl Çalıştırır?

```
▶ class BankaHesabiTest {
2   private BankaHesabi hesap;
3
4   @BeforeEach
5   void hazirla() {
6     hesap = new BankaHesabi(100);
7   }
8
9   @Test
10  void yatirmaBakiyeyiArtirir() {
11    hesap.yatir(50);
12    assertEquals(150.0, hesap.getBakiye(), 0.001);
13  }
14
15  @Test
16  void fazlaCekmeIstisnaFirlatir() {
17    assertThrows(IllegalStateException.class,
18      () -> hesap.cek(500));
19    assertEquals(100.0, hesap.getBakiye(), 0.001);
20  }
21 }
```

heap

test nesnesi #1

`hesap` `null`

JUnit sınıftaki `@Test` metotlarını bulur ve **ilk test için yeni bir test nesnesi** oluşturur. Alan `hesap` henüz `null`.

Adım 1 / 8

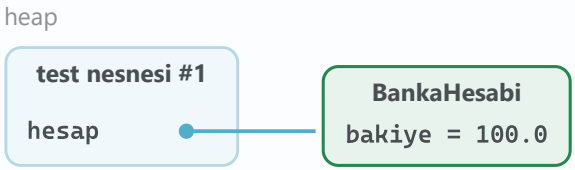
ÇAĞRI YIĞINI

`JUnit`

ÇIKTI

JUnit Testleri Nasıl Çalıştırır?

```
1  class BankaHesabiTest {
2    private BankaHesabi hesap;
3
4    @BeforeEach
5    void hazirla() {
▶      hesap = new BankaHesabi(100);
7    }
8
9    @Test
10   void yatirmaBakiyeyiArtirir() {
11     hesap.yatir(50);
12     assertEquals(150.0, hesap.getBakiye(), 0.001);
13   }
14
15   @Test
16   void fazlaCekmeIstisnaFirlatir() {
17     assertThrows(IllegalStateException.class,
18       () -> hesap.cek(500));
19     assertEquals(100.0, hesap.getBakiye(), 0.001);
20   }
21 }
```



Testten önce `@BeforeEach` çalışır: yeni hesap, bakiye 100.

Adım 2 / 8

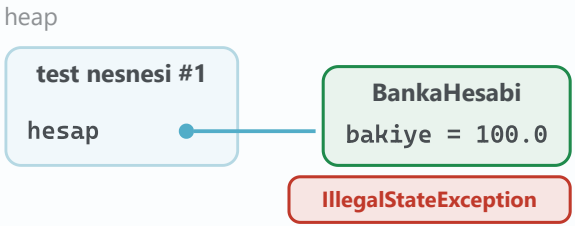
ÇAĞRI YIĞINI

- **hazirla()**
- JUnit

ÇIKTI

JUnit Testleri Nasıl Çalıştırır?

```
1  class BankaHesabiTest {
2    private BankaHesabi hesap;
3
4    @BeforeEach
5    void hazirla() {
6      hesap = new BankaHesabi(100);
7    }
8
9    @Test
10   void yatirmaBakiyeyiArtirir() {
11     hesap.yatir(50);
12     assertEquals(150.0, hesap.getBakiye(), 0.001);
13   }
14
15   @Test
16   void fazlaCekmeIstisnaFirlatir() {
17     assertThrows(IllegalStateException.class,
▶        () -> hesap.cek(500));
19     assertEquals(100.0, hesap.getBakiye(), 0.001);
20   }
21 }
```



Sıra JUnit'tedir: önce `fazlaCekmeIstisnaFirlatir` çalıştı. `assertThrows` lambdayı çağırır: `cek(500)` istisna fırlatır.

Adım 3 / 8

ÇAĞRI YIĞINI

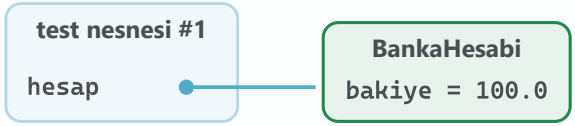
- **`assertThrows(…)`**
- `fazlaCekmeIstisnaFirlatir()`
- `JUnit`

ÇIKTI

JUnit Testleri Nasıl Çalıştırır?

```
1  class BankaHesabiTest {
2    private BankaHesabi hesap;
3
4    @BeforeEach
5    void hazirla() {
6      hesap = new BankaHesabi(100);
7    }
8
9    @Test
10   void yatirmaBakiyeyiArtirir() {
11     hesap.yatir(50);
12     assertEquals(150.0, hesap.getBakiye(), 0.001);
13   }
14
15   @Test
16   void fazlaCekmeIstisnaFirlatir() {
17     assertThrows(IllegalStateException.class,
18       () -> hesap.cek(500));
▶      assertEquals(100.0, hesap.getBakiye(), 0.001);
20   }
21 }
```

heap



✓ test 1 geçti

İstisna beklenen tipte: `assertThrows` geçer. Bakiye hâlâ 100: ikinci doğrulama da geçer.

Adım 4 / 8

ÇAĞRI YIĞINI

`fazlaCekmeIstisnaFirlatir()`

JUnit

ÇIKTI

```
✓ fazlaCekmeIstisnaFirlatir()
```

JUnit Testleri Nasıl Çalıştırır?

```
▶ class BankaHesabiTest {
2   private BankaHesabi hesap;
3
4   @BeforeEach
5   void hazirla() {
6     hesap = new BankaHesabi(100);
7   }
8
9   @Test
10  void yatirmaBakiyeyiArtirir() {
11    hesap.yatir(50);
12    assertEquals(150.0, hesap.getBakiye(), 0.001);
13  }
14
15  @Test
16  void fazlaCekmeIstisnaFirlatir() {
17    assertThrows(IllegalStateException.class,
18      () -> hesap.cek(500));
19    assertEquals(100.0, hesap.getBakiye(), 0.001);
20  }
21 }
```

heap

test nesnesi #2

`hesap` `null`

İkinci test için **yepyeni** bir test nesnesi (#2). Önceki nesne ve hesabı artık kullanılmaz.

ÇAĞRI YIĞINI

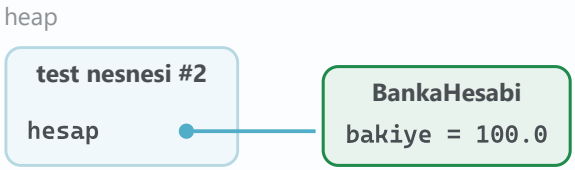
`JUnit`

ÇIKTI

```
✓ fazlaCekmeIstisnaFirlatir()
```

JUnit Testleri Nasıl Çalıştırır?

```
1  class BankaHesabiTest {
2    private BankaHesabi hesap;
3
4    @BeforeEach
5    void hazirla() {
▶      hesap = new BankaHesabi(100);
7    }
8
9    @Test
10   void yatirmaBakiyeyiArtirir() {
11     hesap.yatir(50);
12     assertEquals(150.0, hesap.getBakiye(), 0.001);
13   }
14
15   @Test
16   void fazlaCekmeIstisnaFirlatir() {
17     assertThrows(IllegalStateException.class,
18       () -> hesap.cek(500));
19     assertEquals(100.0, hesap.getBakiye(), 0.001);
20   }
21 }
```



`@BeforeEach` yine çalışır: bu test de bakiye 100 ile başlar.

Adım 6 / 8

ÇAĞRI YIĞINI

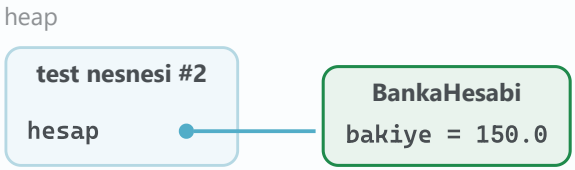
- **hazirla()**
- JUnit

ÇIKTI

```
✓ fazlaCekmeIstisnaFirlatir()
```

JUnit Testleri Nasıl Çalıştırır?

```
1  class BankaHesabiTest {
2    private BankaHesabi hesap;
3
4    @BeforeEach
5    void hazirla() {
6      hesap = new BankaHesabi(100);
7    }
8
9    @Test
10   void yatirmaBakiyeyiArtirir() {
▶      hesap.yatir(50);
12     assertEquals(150.0, hesap.getBakiye(), 0.001);
13   }
14
15   @Test
16   void fazlaCekmeIstisnaFirlatir() {
17     assertThrows(IllegalStateException.class,
18       () -> hesap.cek(500));
19     assertEquals(100.0, hesap.getBakiye(), 0.001);
20   }
21 }
```



Çalıştır: `yatir(50)`, bakiye 150 olur.

Adım 7 / 8

ÇAĞRI YIĞINI

yatirmaBakiyeyiArtirir()

JUnit

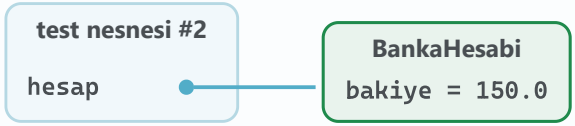
ÇIKTI

```
✓ fazlaCekmeIstisnaFirlatir()
```

JUnit Testleri Nasıl Çalıştırır?

```
1  class BankaHesabiTest {
2    private BankaHesabi hesap;
3
4    @BeforeEach
5    void hazirla() {
6      hesap = new BankaHesabi(100);
7    }
8
9    @Test
10   void yatirmaBakiyeyiArtirir() {
11     hesap.yatir(50);
▶      assertEquals(150.0, hesap.getBakiye(), 0.001);
13   }
14
15   @Test
16   void fazlaCekmeIstisnaFirlatir() {
17     assertThrows(IllegalStateException.class,
18       () -> hesap.cek(500));
19     assertEquals(100.0, hesap.getBakiye(), 0.001);
20   }
21 }
```

heap



✓ test 2 geçti

Doğrula: 150.0 = 150.0 (fark ≤ 0.001). İki test de geçti; ikisi de **birbirinden bağımsız** çalıştı.

Adım 8 / 8

ÇAĞRI YIĞINI

- `yatirmaBakiyeyiArtirir()`
- JUnit

ÇIKTI

```
✓ fazlaCekmeIstisnaFirlatir()
✓ yatirmaBakiyeyiArtirir()
```

İstisnaları Test Etmek: `assertThrows`

```
class BankaIstisnaTest {

  @Test
  void fazlaCekmeMesajiDogru() {
    BankaHesabi hesap = new BankaHesabi(100);
    IllegalStateException e = assertThrows(
      IllegalStateException.class, () -> hesap.cek(500));
    assertEquals("Yetersiz bakiye", e.getMessage());
  }

  @Test
  void negatifYatirmaReddedilir() {
    BankaHesabi hesap = new BankaHesabi(100);
    assertThrows(IllegalArgumentException.class,
      () -> hesap.yatir(-10));
  }

  @Test
  void yanlisBeklenti() {
    BankaHesabi hesap = new BankaHesabi(100);
    assertThrows(IllegalStateException.class,
      () -> hesap.cek(50));
  }
}
```

- `assertThrows` fırlatılan istisna nesnesini **döndürür**; mesajı da doğrulanabilir.
- Hatalı girdinin **reddedildiğini** sınamak, doğru girdiyi sınamak kadar önemlidir.
- Son test bilerek yanlış: 50 TL çekmek geçerlidir, istisna yok.

Test Sonuçları

✓ `fazlaCekmeMesajiDogru()`
✓ `negatifYatirmaReddedilir()`
✗ `yanlisBeklenti()`

```
Expected
java.lang.IllegalStateException to be
thrown, but nothing was thrown.
```

3 test · 2 başarılı · 1 başarısız

JUnit 4'te: `@Test(expected = …class)`.

Test Yaşam Döngüsü Etiketleri

```
class YasamDongusuTest {

  @BeforeAll
  static void basla() { System.out.println("BeforeAll"); }

  @BeforeEach
  void hazirla() { System.out.println("  BeforeEach"); }

  @Test
  void testA() { System.out.println("    testA"); }

  @Test
  void testB() { System.out.println("    testB"); }

  @AfterEach
  void temizle() { System.out.println("  AfterEach"); }

  @AfterAll
  static void bitir() { System.out.println("AfterAll"); }
}
```

Çıktı

```
BeforeAll
  BeforeEach
    testA
  AfterEach
  BeforeEach
    testB
  AfterEach
AfterAll
```

Etiket	Ne zaman çalışır?
`@BeforeAll`	Tüm testlerden önce **bir kez** (`static`)
`@BeforeEach`	Her testten önce
`@AfterEach`	Her testten sonra (test kalsa bile)
`@AfterAll`	Tüm testlerden sonra bir kez (`static`)

Testlerin **kendi aralarındaki sırası** belirli ama kod sırası değildir: testler birbirine bağımlı yazılmamalıdır.

Başarısız Test: Kırmızıdan Yeşile

`DiziIslem` (İLK SÜRÜM)

```
public class DiziIslem {
  public static int enBuyuk(int[] dizi) {
    int enb = 0;
    for (int i = 0; i < dizi.length; i++) {
      if (dizi[i] > enb) {
        enb = dizi[i];
      }
    }
    return enb;
  }
}
```

`DiziIslemTest`

```
class DiziIslemTest {

  @Test
  void pozitifDizi() {
    int[] dizi = {3, 9, 4};
    assertEquals(9, DiziIslem.enBuyuk(dizi));
  }

  @Test
  void tumuNegatifDizi() {
    int[] dizi = {-5, -2, -8};
    assertEquals(-2, DiziIslem.enBuyuk(dizi));
  }
}
```

Kod derleniyor, ilk bakışta doğru görünüyor. İki test yazalım: biri pozitif, biri tamamen negatif bir dizi.

Başarısız Test: Kırmızıdan Yeşile

`DiziIslem` (İLK SÜRÜM)

```
public class DiziIslem {
  public static int enBuyuk(int[] dizi) {
    int enb = 0;
    for (int i = 0; i < dizi.length; i++) {
      if (dizi[i] > enb) {
        enb = dizi[i];
      }
    }
    return enb;
  }
}
```

Test Sonuçları

✓ `pozitifDizi()`

✗ `tumuNegatifDizi()`

```
org.opentest4j.AssertionFailedError: expected:
<-2> but was: <0>
```

2 test · 1 başarılı · **1 başarısız**

expected = beklenen (-2), **but was** = gerçekte dönen (0). 0 dizide yok bile!

Kırmızı: `enb = 0` başlangıcı, tüm öğeler negatifken hiç aşılmıyor.

Başarısız Test: Kırmızıdan Yeşile

`DiziIslem` (DÜZELTİLMİŞ)

```
public class DiziIslem {
  public static int enBuyuk(int[] dizi) {
    int enb = dizi[0];                // ilk eleman
    for (int i = 1; i < dizi.length; i++) {
      if (dizi[i] > enb) {
        enb = dizi[i];
      }
    }
    return enb;
  }
}
```

Test Sonuçları

✓ `pozitifDizi()`

✓ `tumuNegatifDizi()`

2 test · 2 başarılı

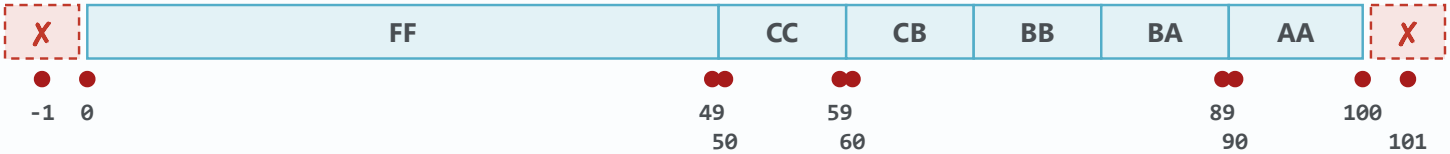
En büyük değer dizinin **ilk öğesiyle** başlatılır. Bu test artık kalıcıdır: hata geri gelirse hemen yakalanır.

Yeşil: düzeltmeden sonra aynı testler yeniden çalıştırıldı, ikisi de geçti.

Hangi Değerleri Test Etmeli? Sınır Değerleri

```java
public class Not {
  public static String harfNotu(int puan) {
    if (puan < 0 || puan > 100) {
      throw new IllegalArgumentException("Geçersiz puan: " + puan);
    }
    if (puan >= 90) return "AA";
    if (puan >= 80) return "BA";
    if (puan >= 70) return "BB";
    if (puan >= 60) return "CB";
    if (puan >= 50) return "CC";
    return "FF";
  }
}
```

- Hatalar en çok **sınırlarda** olur: `>` yerine `>=`, `< 100` yerine `<= 100`...
- **Sınır değer analizi**: her aralığın iki ucu (49 / 50, 89 / 90) ve geçersiz bölgenin ilk değeri (−1, 101) seçilir.
- **Denklik sınıfları**: aynı aralıktaki 61, 65, 68 aynı sonucu verir; bir tanesi yeterlidir.



● test edilen değer: her aralığın iki ucu ve geçersiz bölgenin ilk değeri

Parametreli Testler JUnit 5

```java
class NotTest {

  @ParameterizedTest
  @CsvSource({"0, FF", "49, FF", "50, CC", "59, CC",
        "60, CB", "89, BA", "90, AA", "100, AA"})
  void sinirDegerleri(int puan, String beklenen) {
    assertEquals(beklenen, Not.harfNotu(puan));
  }

  @Test
  void aralikDisiPuanReddedilir() {
    assertThrows(IllegalArgumentException.class,
      () -> Not.harfNotu(-1));
    assertThrows(IllegalArgumentException.class,
      () -> Not.harfNotu(101));
  }
}
```

- `@ParameterizedTest` : aynı test metodu **farklı verilerle** tekrar tekrar çalışır.
- `@CsvSource` : her dizgi bir satır; virgülle ayrılan değerler sırayla parametrelere (`puan` , `beklenen`) dönüştürülür.
- Bağımlılık: `junit-jupiter-params` (`junit-jupiter` paketine dahildir).

sinirDegerleri(int, String)

- ✓ `[1] 0, FF`
- ✓ `[2] 49, FF`
- ✓ `[3] 50, CC`
- ✓ `[4] 59, CC`
- ✓ `[5] 60, CB`
- ✓ `[6] 89, BA`
- ✓ `[7] 90, AA`
- ✓ `[8] 100, AA`
- ✓ `aralikDisiPuanReddedilir()`

9 test · 9 başarılı

Parametreli Test Adım Adım

```
1 class NotTest {
▶   @ParameterizedTest
▶   @CsvSource({"0, FF", "49, FF", "50, CC", "59,
  CC",
▶          "60, CB", "89, BA", "90, AA", "100, AA"})
5   void sinirDegerleri(int puan, String beklenen) {
6     assertEquals(beklenen, Not.harfNotu(puan));
7   }
8 }
```



her aralığın iki ucu sınanıyor

JUnit `@CsvSource`'taki 8 dizgiyi okur; her biri ayrı bir test çağrısı olacak.

Adım 1 / 9

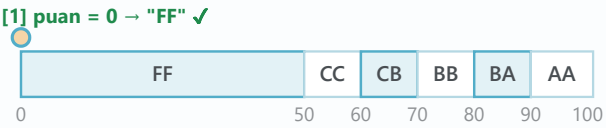
DEĞİŞKENLER

`satır sayısı = 8`

ÇIKTI

Parametreli Test Adım Adım

```
1 class NotTest {
2   @ParameterizedTest
3   @CsvSource({"0, FF", "49, FF", "50, CC", "59,
  CC",
4           "60, CB", "89, BA", "90, AA", "100, AA"})
5   void sinirDegerleri(int puan, String beklenen) {
▶       assertEquals(beklenen, Not.harfNotu(puan));
7   }
8 }
```



her aralığın iki ucu sınanıyor

Satır 1: `"0, FF"` virgülden bölünür; `"0"` `int` 'e çevrilip `puan` 'a, `"FF"` `beklenen` 'e verilir.

Adım 2 / 9

DEĞİŞKENLER

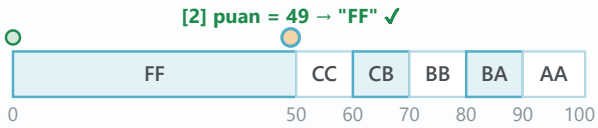
`puan = 0` `beklenen = "FF"` `gerçek = "FF"`

ÇIKTI

```
✓ [1] 0, FF
```

Parametreli Test Adım Adım

```
1 class NotTest {
2   @ParameterizedTest
3   @CsvSource({"0, FF", "49, FF", "50, CC", "59,
  CC",
4         "60, CB", "89, BA", "90, AA", "100, AA"})
5   void sinirDegerleri(int puan, String beklenen) {
▶     assertEquals(beklenen, Not.harfNotu(puan));
7   }
8 }
```



her aralığın iki ucu sınanıyor

49: FF aralığının **üst ucu**. `>= 50` yerine yanlışlıkla `> 48` yazılsaydı burada kalırdı.

Adım 3 / 9

DEĞİŞKENLER

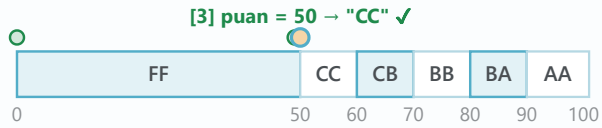
`puan = 49` `beklenen = "FF"` `gerçek = "FF"`

ÇIKTI

```
✓ [1] 0, FF
✓ [2] 49, FF
```

Parametreli Test Adım Adım

```
1 class NotTest {
2   @ParameterizedTest
3   @CsvSource({"0, FF", "49, FF", "50, CC", "59, CC",
4         "60, CB", "89, BA", "90, AA", "100, AA"})
5   void sinirDegerleri(int puan, String beklenen) {
▶     assertEquals(beklenen, Not.harfNotu(puan));
7   }
8 }
```



her aralığın iki ucu sınanıyor

50: CC'nin **alt ucu**; 49 ile birlikte aradaki sınırı iki yandan sınar.

Adım 4 / 9

DEĞİŞKENLER

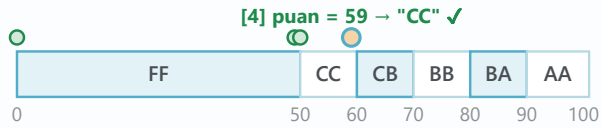
`puan = 50` `beklenen = "CC"` `gerçek = "CC"`

ÇIKTI

```
✓ [1] 0, FF
✓ [2] 49, FF
✓ [3] 50, CC
```

Parametreli Test Adım Adım

```
1 class NotTest {
2   @ParameterizedTest
3   @CsvSource({"0, FF", "49, FF", "50, CC", "59,
  CC",
4           "60, CB", "89, BA", "90, AA", "100, AA"})
5   void sinirDegerleri(int puan, String beklenen) {
▶       assertEquals(beklenen, Not.harfNotu(puan));
7   }
8 }
```



her aralığın iki ucu sınanıyor

59: CC'nin üst ucu.

Adım 5 / 9

DEĞİŞKENLER

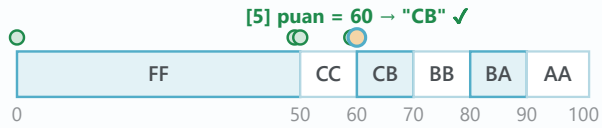
`puan = 59` `beklenen = "CC"` `gerçek = "CC"`

ÇIKTI

```
✓ [1] 0, FF
✓ [2] 49, FF
✓ [3] 50, CC
✓ [4] 59, CC
```

Parametreli Test Adım Adım

```
1 class NotTest {
2   @ParameterizedTest
3   @CsvSource({"0, FF", "49, FF", "50, CC", "59,
  CC",
4         "60, CB", "89, BA", "90, AA", "100, AA"})
5   void sinirDegerleri(int puan, String beklenen) {
▶     assertEquals(beklenen, Not.harfNotu(puan));
7   }
8 }
```



her aralığın iki ucu sınanıyor

60: CB'nin alt ucu.

Adım 6 / 9

DEĞİŞKENLER

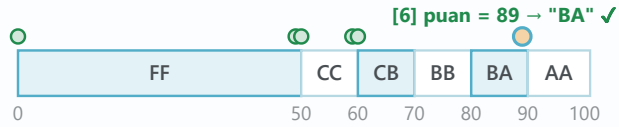
`puan = 60` `beklenen = "CB"` `gerçek = "CB"`

ÇIKTI

```
✓ [1] 0, FF
✓ [2] 49, FF
✓ [3] 50, CC
✓ [4] 59, CC
✓ [5] 60, CB
```

Parametreli Test Adım Adım

```
1 class NotTest {
2   @ParameterizedTest
3   @CsvSource({"0, FF", "49, FF", "50, CC", "59,
  CC",
4           "60, CB", "89, BA", "90, AA", "100, AA"})
5   void sinirDegerleri(int puan, String beklenen) {
▶       assertEquals(beklenen, Not.harfNotu(puan));
7   }
8 }
```



her aralığın iki ucu sınanıyor

89: BA'nın üst ucu.

Adım 7 / 9

DEĞİŞKENLER

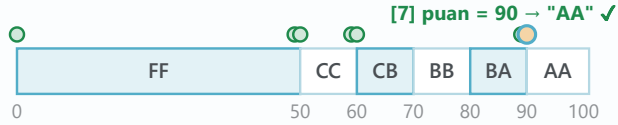
`puan = 89` `beklenen = "BA"` `gerçek = "BA"`

ÇIKTI

```
✓ [1] 0, FF
✓ [2] 49, FF
✓ [3] 50, CC
✓ [4] 59, CC
✓ [5] 60, CB
✓ [6] 89, BA
```

Parametreli Test Adım Adım

```
1 class NotTest {
2   @ParameterizedTest
3   @CsvSource({"0, FF", "49, FF", "50, CC", "59,
  CC",
4         "60, CB", "89, BA", "90, AA", "100, AA"})
5   void sinirDegerleri(int puan, String beklenen) {
▶     assertEquals(beklenen, Not.harfNotu(puan));
7   }
8 }
```



her aralığın iki ucu sınanıyor

90: AA'nın alt ucu.

Adım 8 / 9

DEĞİŞKENLER

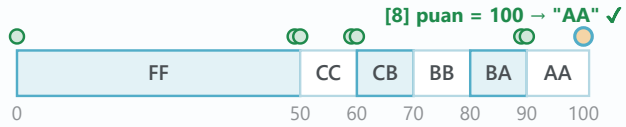
`puan = 90` `beklenen = "AA"` `gerçek = "AA"`

ÇIKTI

```
✓ [1] 0, FF
✓ [2] 49, FF
✓ [3] 50, CC
✓ [4] 59, CC
✓ [5] 60, CB
✓ [6] 89, BA
✓ [7] 90, AA
```

Parametreli Test Adım Adım

```
1 class NotTest {
2   @ParameterizedTest
3   @CsvSource({"0, FF", "49, FF", "50, CC", "59,
  CC",
4           "60, CB", "89, BA", "90, AA", "100, AA"})
5   void sinirDegerleri(int puan, String beklenen) {
▶       assertEquals(beklenen, Not.harfNotu(puan));
7   }
8 }
```



her aralığın iki ucu sınanıyor

100: geçerli bölgenin son değeri; `puan > 100` kontrolü doğruysa AA döner.

Adım 9 / 9

DEĞİŞKENLER

`puan = 100` `beklenen = "AA"` `gerçek = "AA"`

ÇIKTI

```
✓ [1] 0, FF
✓ [2] 49, FF
✓ [3] 50, CC
✓ [4] 59, CC
✓ [5] 60, CB
✓ [6] 89, BA
✓ [7] 90, AA
✓ [8] 100, AA
```

İyi Bir Birim Testinin Özellikleri: FIRST

F · Hızlı (Fast)
Milisaniyeler içinde biter; binlercesi her değişiklikten sonra çalıştırılabilir. Dosya, ağ, veri tabanı yok.

I · Bağımsız (Independent)
Başka bir teste, test sırasına veya önceki testin bıraktığı duruma bağlı değildir.

R · Tekrarlanabilir (Repeatable)
Her makinede, her zaman aynı sonucu verir: rastgelelik ve saat kontrol altında.

S · Kendini doğrulayan (Self-validating)
Sonuç ✓ ya da ✗; çıktıyı bir insanın okumasına gerek yok.

T · Zamanında (Timely)
Kodla birlikte (hatta koddan önce) yazılır; aylar sonraya bırakılmaz.

Ve okunabilir
Açıklayıcı ad, AAA düzeni, test başına tek davranış. Test, kodun nasıl kullanılacağını gösteren bir belgedir.

Birim Testi mi, Tümleştirme Testi mi?

	Birim testi	Tümleştirme (entegrasyon) testi
Kapsam	Tek sınıf / metot, **yalıtılmış**	Birlikte çalışan birkaç bileşen
Hız	Milisaniyeler	Saniyeler (veri tabanı, dosya, ağ)
Dış bağımlılık	Yerine basit bir sahte nesne (test dublörü) konur	Gerçek veri tabanı, dosya, servis
Bulduğu hata	Yerel mantık hatası; yeri hemen belli	Uyumsuz arayüz, yanlış yapılandırma

DAVRANIŞI ANLATAN TEST ADLARI

```
@Test void test1() { … }
@Test void altiUrundeIndirimVar() { … }
@Test void besUrundeIndirimYok() { … }
```

- Gerçeklemenin iç ayrıntısını değil, **dışarıdan görünen davranışı** sınanır.
- Birim testinde `Thread.sleep`, gerçek ağ ve dosya kullanılmaz: test yavaşlar ve rastgele kalır.
- Testler arasında değişen durum (ör. `static` liste) paylaşılmaz.

Regresyon Testi (Regression Testing)

- Yapılan değişikliklerden sonra **önceden çalışan her şeyin hâlâ çalıştığını** doğrulamak anlamına gelir. Regresyon = "geriye gidiş": düzelen bir şeyin yeniden bozulması.
- Yöntem: daha önce yazılmış **tüm testler** her değişiklikten sonra yeniden çalıştırılır.



Belirleyici (deterministic) olmalıdır

Testler, aynı koşullarda **her seferinde aynı sonucu** vermelidir. Bazen geçip bazen kalan ("kararsız", flaky) test güveni yok eder.

Tekrarlanabilir (repeatable) olmalıdır

Testler farklı zamanlarda, farklı makinelerde ve durumlarda **tekrar edilebilir** şekilde tasarlanmalıdır.

Rastgelelik İçeren Kodu Belirleyici Test Etmek

```java
import java.util.Random;

public class Zar {
  private final Random rastgele;

  public Zar(Random rastgele) {      // dışarıdan verilir
    this.rastgele = rastgele;
  }

  public int at() {
    return rastgele.nextInt(6) + 1;
  }
}
```

AYNI TOHUMLA (42) BEŞ ATIŞ

Konsol

```
3 4 1 3 1
```

```java
class ZarTest {

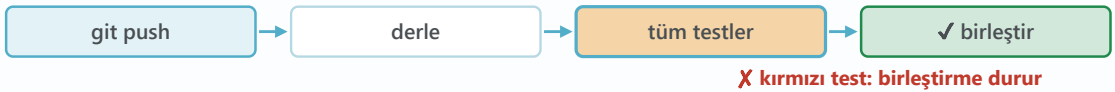
  @Test
  void ayniTohumAyniSonuclar() {
    Zar z1 = new Zar(new Random(42));
    Zar z2 = new Zar(new Random(42));
    for (int i = 0; i < 100; i++) {
      assertEquals(z1.at(), z2.at());
    }
  }
}
```

`Random` nesnesi **dışarıdan verilir** (bağımlılık enjeksiyonu). Test, sabit **tohum** (seed) vererek her çalıştırmada aynı sayıları alır; zar sınıfı içinde `new Random()` yazılsaydı sonuçlar her seferinde değişirdi.

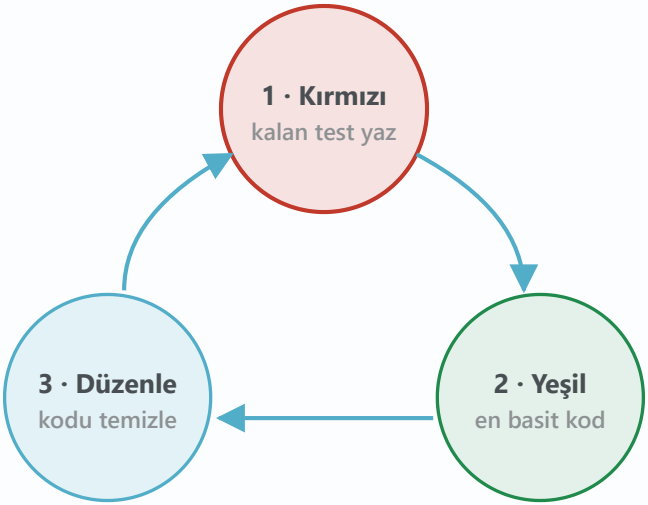
Otomatik Testler

- Kodu **manuel olarak** test etmek yerine, **otomatik testler** yazılmalıdır.
- Harcanan çaba aşağı yukarı **aynıdır**, ancak sonunda geriye otomatik bir test kalır.
- Bu test, **istenildiği zaman** (her derlemede, her değişiklikte) tekrar çalıştırılabilir.
- Yazılım bir kez doğru çalıştı diye bir daha test etmenin **gereksiz** olduğunu düşünmek **yanlıştır**: kod değişir, kütüphaneler değişir, ortam değişir.
- **Sürekli entegrasyon** (CI, ör. GitHub Actions, Jenkins): her `git push` sonrası tüm testleri sunucuda otomatik çalıştırır; kırmızı test birleştirmeyi durdurur.

	Elle	Otomatik
İlk maliyet	Düşük	Biraz yüksek
Tekrar maliyeti	**Her seferinde aynı**	**≈ 0**
Hız	Dakikalar	Saniyeler
Güvenilirlik	İnsan hatası	Her seferinde aynı
Regresyon	Pratikte yapılmaz	Her değişiklikte
Belge değeri	Yok	Kullanım örneği



Test Güdümlü Geliştirme (TDD)



- **Test-Driven Development**: testi **koddan önce** yazmak.
- **Kırmızı**: henüz olmayan davranış için bir test yazılır; test kalır (ya da derlenmez).
- **Yeşil**: testi geçirecek **en basit** kod yazılır.
- **Yeniden düzenle** (refactor): kod temizlenir; testler yeşil kaldıkça davranış korunur.
- Örnek: önce `assertEquals("CC", Not.harfNotu(50))` testi yazılır, sonra `harfNotu` metodu. Her yeni not aralığı yeni bir döngüdür.

Kazanç: her satır kod bir testle doğar; regresyon testleri kendiliğinden birikir.

Hata Ayıklayıcı (Debugger)

Hata ayıklayıcı, **çalışan bir programın durumunu** incelemeye olanak sağlar. Hata ayıklayıcı ile yapılabilecekler:

- Programı **komut komut** (satır satır) çalıştırma
- Çalışan programın **kaynak kodunu** o anki satırla birlikte inceleme
- **Değişkenlerin değerlerini** görüntüleme ve gerektiğinde değiştirebilme
- Programa **kesme noktaları** (breakpoint) ekleyebilme
- Programı belirtilen bir kesme noktasına **kadar çalıştırma**
- Çökmüş ya da durdurulmuş programın durumunu bir **döküm dosyası** üzerinden inceleme (C'de *core file*; Java'da heap dökümü `.hprof` ve iş parçacığı dökümü)

IDE hata ayıklayıcıları

IntelliJ IDEA, Eclipse, NetBeans, VS Code: satır numarasının yanına tıklayarak kesme noktası, **Debug** düğmesiyle başlatma.

println ile ayıklama

Hızlıdır ama kodu kirletir, yeniden derleme ister ve sonradan silinmesi unutulur. Hata ayıklayıcı kodu değiştirmez.



HATA AYIKLAMA

Kesme Noktaları ve Adım Komutları

Komut	Ne yapar?	IntelliJ	Eclipse
Kesme noktası (breakpoint)	Program bu satıra gelince, satır **çalışmadan önce** durur	Ctrl+F8	Ctrl+Shift+B
Step Over (üzerinden geç)	Satırı çalıştırır; metot çağrısı varsa **içine girmeden** tamamlar	F8	F6
Step Into (içine gir)	Satırdaki metot çağrısının **içine** girer	F7	F5
Step Out (dışarı çık)	Bulunulan metodu bitirir, **çağıran** metoda döner	Shift+F8	F7
Resume (devam)	Bir sonraki kesme noktasına kadar çalışır	F9	F8
Evaluate / Watch	Bir ifadenin (ör. `dizi[i] > enb`) değerini anlık hesaplar, sürekli izler	Alt+F8	Ctrl+Shift+I

Koşullu kesme noktası
Yalnızca koşul doğruysa durur, ör. `i == 999` : binlerce turluk döngüde tam aranan anı yakalar.

İstisna kesme noktası
Belirli bir istisna (ör. `NullPointerException`) fırlatıldığı anda durur.

Kısayollar varsayılan tuş eşlemesine göredir; menüden de (Run / Debug) aynı komutlara ulaşılır.

HATA AYIKLAMA · ADIM ADIM

Hata Ayıklayıcı ile Kusuru Bulmak

```
1  public class HataAyikla {
2    public static int enBuyuk(int[] dizi) {
3      int enb = 0;
4      for (int i = 0; i < dizi.length; i++) {
5        if (dizi[i] > enb) {
6          enb = dizi[i];
7        }
8      }
9      return enb;
10   }
11
12   public static void main(String[] args) {
▶      int[] sicaklik = {-5, -2, -8};
14     int sonuc = enBuyuk(sicaklik);
15     System.out.println("En yüksek: " + sonuc);
16   }
17 }
```

dizi

[0]	[1]	[2]
−5	−2	−8

Kesme noktası 5. satırda (`if`). Program **Debug** ile başlatıldı.

Adım 1 / 8

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Hata Ayıklayıcı ile Kusuru Bulmak

```
1  public class HataAyikla {
2    public static int enBuyuk(int[] dizi) {
3      int enb = 0;
4      for (int i = 0; i < dizi.length; i++) {
5        if (dizi[i] > enb) {
6          enb = dizi[i];
7        }
8      }
9      return enb;
10   }
11
12   public static void main(String[] args) {
13     int[] sicaklik = {-5, -2, -8};
▶    int sonuc = enBuyuk(sicaklik);
15     System.out.println("En yüksek: " + sonuc);
16   }
17 }
```

dizi

[0]	[1]	[2]
−5	−2	−8

Step Into: `enBuyuk` çağrısının içine giriyoruz.

Adım 2 / 8

DEĞİŞKENLER

`sicaklik = [-5, -2, -8]`

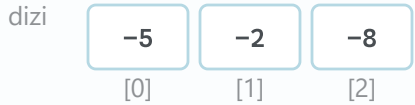
ÇAĞRI YIĞINI

`main()`

ÇIKTI

Hata Ayıklayıcı ile Kusuru Bulmak

```
1  public class HataAyikla {
2    public static int enBuyuk(int[] dizi) {
▶      int enb = 0;
4      for (int i = 0; i < dizi.length; i++) {
5        if (dizi[i] > enb) {
6          enb = dizi[i];
7        }
8      }
9      return enb;
10   }
11
12   public static void main(String[] args) {
13     int[] sicaklik = {-5, -2, -8};
14     int sonuc = enBuyuk(sicaklik);
15     System.out.println("En yüksek: " + sonuc);
16   }
17 }
```



Yeni çerçeve: `dizi` aynı diziyi gösterir. `enb` henüz atanmadı.

Adım 3 / 8

DEĞİŞKENLER

`dizi = [-5, -2, -8]`

ÇAĞRI YIĞINI

`enBuyuk(dizi)`

`main()`

ÇIKTI

Hata Ayıklayıcı ile Kusuru Bulmak

```
1  public class HataAyikla {
2    public static int enBuyuk(int[] dizi) {
3      int enb = 0;
4      for (int i = 0; i < dizi.length; i++) {
▶        if (dizi[i] > enb) {
6          enb = dizi[i];
7        }
8      }
9      return enb;
10   }
11
12   public static void main(String[] args) {
13     int[] sicaklik = {-5, -2, -8};
14     int sonuc = enBuyuk(sicaklik);
15     System.out.println("En yüksek: " + sonuc);
16   }
17 }
```



Kesme noktası: `enb = 0`, `dizi[0] = -5`.
İzlenen ifade **false**.

Adım 4 / 8

DEĞİŞKENLER

`dizi = [-5, -2, -8]` `enb = 0` `i = 0`

ÇAĞRI YIĞINI

`enBuyuk(dizi)`

`main()`

ÇIKTI

Hata Ayıklayıcı ile Kusuru Bulmak

```
1  public class HataAyikla {
2    public static int enBuyuk(int[] dizi) {
3      int enb = 0;
4      for (int i = 0; i < dizi.length; i++) {
▶        if (dizi[i] > enb) {
6          enb = dizi[i];
7        }
8      }
9      return enb;
10   }
11
12   public static void main(String[] args) {
13     int[] sicaklik = {-5, -2, -8};
14     int sonuc = enBuyuk(sicaklik);
15     System.out.println("En yüksek: " + sonuc);
16   }
17 }
```



`izle: dizi[i] > enb → false`

Resume: `i = 1` , `-2 > 0` yine false; `enb` değişmedi.

Adım 5 / 8

DEĞİŞKENLER

`dizi = [-5, -2, -8]` `enb = 0` `i = 1`

ÇAĞRI YIĞINI

`enBuyuk(dizi)`

`main()`

ÇIKTI

Hata Ayıklayıcı ile Kusuru Bulmak

```
1  public class HataAyikla {
2    public static int enBuyuk(int[] dizi) {
3      int enb = 0;
4      for (int i = 0; i < dizi.length; i++) {
▶        if (dizi[i] > enb) {
6          enb = dizi[i];
7        }
8      }
9      return enb;
10   }
11
12   public static void main(String[] args) {
13     int[] sicaklik = {-5, -2, -8};
14     int sonuc = enBuyuk(sicaklik);
15     System.out.println("En yüksek: " + sonuc);
16   }
17 }
```



`i = 2`, `-8 > 0` false. Şüphe: `enb` hiç güncellenmiyor!

Adım 6 / 8

DEĞİŞKENLER

`dizi = [-5, -2, -8]` `enb = 0` `i = 2`

ÇAĞRI YIĞINI

`enBuyuk(dizi)`

`main()`

ÇIKTI

Hata Ayıklayıcı ile Kusuru Bulmak

```
1  public class HataAyikla {
2    public static int enBuyuk(int[] dizi) {
3      int enb = 0;
4      for (int i = 0; i < dizi.length; i++) {
5        if (dizi[i] > enb) {
6          enb = dizi[i];
7        }
8      }
▶      return enb;
10   }
11
12   public static void main(String[] args) {
13     int[] sicaklik = {-5, -2, -8};
14     int sonuc = enBuyuk(sicaklik);
15     System.out.println("En yüksek: " + sonuc);
16   }
17 }
```



Döngü bitti, 0 döndürülüyor. **Kusur bulundu**: başlangıç değeri 0, tüm öğeler negatifse yanlış. Doğrusu `enb = dizi[0]`.

Adım 7 / 8

DEĞİŞKENLER

`dizi = [-5, -2, -8]` `enb = 0`

ÇAĞRI YIĞINI

`enBuyuk(dizi)`

`main()`

ÇIKTI

Hata Ayıklayıcı ile Kusuru Bulmak

```
1  public class HataAyikla {
2    public static int enBuyuk(int[] dizi) {
3      int enb = 0;
4      for (int i = 0; i < dizi.length; i++) {
5        if (dizi[i] > enb) {
6          enb = dizi[i];
7        }
8      }
9      return enb;
10   }
11
12   public static void main(String[] args) {
13     int[] sicaklik = {-5, -2, -8};
14     int sonuc = enBuyuk(sicaklik);
▶      System.out.println("En yüksek: " + sonuc);
16   }
17 }
```

dizi

[0]	[1]	[2]
−5	−2	−8

Arıza ekranda: `En yüksek: 0`. Düzeltmeden sonra bu durum bir **birim testi** olarak eklenir.

Adım 8 / 8

DEĞİŞKENLER

`sicaklik = [-5, -2, -8]` `sonuc = 0`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
En yüksek: 0
```

Step Into, Step Over, Step Out

```
1  public class AdimOrnek {
2    static int kare(int x) {
3      return x * x;
4    }
5
6    static int kareToplam(int a, int b) {
7      int k1 = kare(a);
8      int k2 = kare(b);
9      return k1 + k2;
10   }
11
12   public static void main(String[] args) {
▶      int s = kareToplam(3, 4);
14     System.out.println("s = " + s);
15   }
16 }
```

Kesme noktasında durdu Debug

Kesme noktası 13. satırda: satır **henüz çalışmadı**.

Adım 1 / 9

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

HATA AYIKLAMA · ADIM ADIM

Step Into, Step Over, Step Out

```
1  public class AdimOrnek {
2    static int kare(int x) {
3      return x * x;
4    }
5
6    static int kareToplam(int a, int b) {
▶      int k1 = kare(a);
8      int k2 = kare(b);
9      return k1 + k2;
10   }
11
12   public static void main(String[] args) {
13     int s = kareToplam(3, 4);
14     System.out.println("s = " + s);
15   }
16 }
```

Step Into F7

Step Into: `kareToplam` 'ın içine girildi; yığına yeni çerçeve eklendi.

Adım 2 / 9

DEĞİŞKENLER

`a = 3` `b = 4`

ÇAĞRI YIĞINI

`kareToplam(3, 4)`

`main()`

ÇIKTI

Step Into, Step Over, Step Out

```
1  public class AdimOrnek {
2    static int kare(int x) {
3      return x * x;
4    }
5
6    static int kareToplam(int a, int b) {
7      int k1 = kare(a);
▶      int k2 = kare(b);
9      return k1 + k2;
10   }
11
12   public static void main(String[] args) {
13     int s = kareToplam(3, 4);
14     System.out.println("s = " + s);
15   }
16 }
```

Step Over F8

Step Over: `kare(3)` çalıştı ama **içine girilmedi**; `k1 = 9`.

Adım 3 / 9

DEĞİŞKENLER

`a = 3` `b = 4` `k1 = 9`

ÇAĞRI YIĞINI

`kareToplam(3, 4)`

`main()`

ÇIKTI

Step Into, Step Over, Step Out

```
1  public class AdimOrnek {
2    static int kare(int x) {
▶      return x * x;
4    }
5
6    static int kareToplam(int a, int b) {
7      int k1 = kare(a);
8      int k2 = kare(b);
9      return k1 + k2;
10   }
11
12   public static void main(String[] args) {
13     int s = kareToplam(3, 4);
14     System.out.println("s = " + s);
15   }
16 }
```

Step Into F7

Step Into: bu kez `kare(4)` 'ün içindeyiz; yalnız `x` görünür.

Adım 4 / 9

DEĞİŞKENLER

`x = 4`

ÇAĞRI YIĞINI

- `kare(4)`
- `kareToplam(3, 4)`
- `main()`

ÇIKTI

Step Into, Step Over, Step Out

```
1  public class AdimOrnek {
2    static int kare(int x) {
3      return x * x;
4    }
5
6    static int kareToplam(int a, int b) {
7      int k1 = kare(a);
▶      int k2 = kare(b);
9      return k1 + k2;
10   }
11
12   public static void main(String[] args) {
13     int s = kareToplam(3, 4);
14     System.out.println("s = " + s);
15   }
16 }
```

Step Out `Shift+F8`

Step Out: `kare` bitti, 16 döndü; çağıran satıra dönüldü, atama tamamlanmak üzere.

Adım 5 / 9

DEĞİŞKENLER

`a = 3` `b = 4` `k1 = 9`

ÇAĞRI YIĞINI

`kareToplam(3, 4)`

`main()`

ÇIKTI

Step Into, Step Over, Step Out

```
1  public class AdimOrnek {
2    static int kare(int x) {
3      return x * x;
4    }
5
6    static int kareToplam(int a, int b) {
7      int k1 = kare(a);
8      int k2 = kare(b);
▶      return k1 + k2;
10   }
11
12   public static void main(String[] args) {
13     int s = kareToplam(3, 4);
14     System.out.println("s = " + s);
15   }
16 }
```

Step Over F8

Step Over: `k2 = 16` .

Adım 6 / 9

DEĞİŞKENLER

`a = 3` `b = 4` `k1 = 9` `k2 = 16`

ÇAĞRI YIĞINI

`kareToplam(3, 4)`

`main()`

ÇIKTI

Step Into, Step Over, Step Out

```
1  public class AdimOrnek {
2    static int kare(int x) {
3      return x * x;
4    }
5
6    static int kareToplam(int a, int b) {
7      int k1 = kare(a);
8      int k2 = kare(b);
9      return k1 + k2;
10   }
11
12   public static void main(String[] args) {
▶      int s = kareToplam(3, 4);
14     System.out.println("s = " + s);
15   }
16 }
```

Step Out `Shift+F8`

Step Out: `kareToplam` 25 döndürdü; `main` 'e dönüldü.

Adım 7 / 9

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

HATA AYIKLAMA · ADIM ADIM

Step Into, Step Over, Step Out

```
1  public class AdimOrnek {
2    static int kare(int x) {
3      return x * x;
4    }
5
6    static int kareToplam(int a, int b) {
7      int k1 = kare(a);
8      int k2 = kare(b);
9      return k1 + k2;
10   }
11
12   public static void main(String[] args) {
13     int s = kareToplam(3, 4);
▶      System.out.println("s = " + s);
15   }
16 }
```

Step Over F8

Step Over: `s = 25` .

Adım 8 / 9

DEĞİŞKENLER

`s = 25`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Step Into, Step Over, Step Out

```
1  public class AdimOrnek {
2    static int kare(int x) {
3      return x * x;
4    }
5
6    static int kareToplam(int a, int b) {
7      int k1 = kare(a);
8      int k2 = kare(b);
9      return k1 + k2;
10   }
11
12   public static void main(String[] args) {
13     int s = kareToplam(3, 4);
14     System.out.println("s = " + s);
▶    }
16 }
```

Step Over F8

Step Over: çıktı yazıldı. **Resume** ile program sonlanır.

Adım 9 / 9

DEĞİŞKENLER

`s = 25`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
s = 25
```

Komut Satırında Hata Ayıklama: `jdb`

```java
public class Jdb {
  static int kare(int x) {
    int sonuc = x * x;
    return sonuc;
  }

  public static void main(String[] args) {
    int toplam = 0;
    for (int i = 1; i <= 3; i++) {
      toplam += kare(i);
    }
    System.out.println("Toplam: " + toplam);
  }
}
```

DERLEME VE HATA AYIKLAYICIYLA BAŞLATMA

```
javac -g Jdb.java
jdb Jdb
```

`-g` : yerel değişken adlarını da sınıf dosyasına yazar; `locals` komutu bunlara ihtiyaç duyar.

jdb (Java Debugger), JDK ile gelen **komut satırı** hata ayıklayıcısıdır; IDE'lerin yaptığını komutlarla yapar.

Komut	Anlamı
`stop at Sınıf:satır`	kesme noktası koyar
`clear Sınıf:satır`	kesme noktasını kaldırır
`run` / `cont`	başlatır / devam eder
`step` / `next` / `step up`	Step Into / Over / Out
`where`	çağrı yığını
`locals` , `print x`	yerel değişkenler, bir değer

Çalışan bir uygulamaya bağlanmak için uygulama `java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=5005 Uygulama` ile başlatılır; IDE ya da `jdb -attach 5005` bağlanır.

Bir `jdb` Oturumu

jdb Jdb (1)

```
> stop at Jdb:4
Deferring breakpoint Jdb:4.
It will be set after the class is loaded.
> run
VM Started: Set deferred breakpoint Jdb:4
Breakpoint hit: "thread=main", Jdb.kare(),
line=4 bci=4
main[1] where
  [1] Jdb.kare (Jdb.java:4)
  [2] Jdb.main (Jdb.java:10)
main[1] locals
Method arguments:
x = 1
Local variables:
sonuc = 1
```

jdb Jdb (2)

```
main[1] cont
Breakpoint hit: "thread=main", Jdb.kare(),
line=4 bci=4
main[1] print sonuc
 sonuc = 4
main[1] clear Jdb:4
Removed: breakpoint Jdb:4
main[1] cont
> Toplam: 14
```

- Kesme noktası 4. satırda (`return sonuc;`): program bu satırı çalıştırmadan durur.
- `where` : `main` (10. satır) → `kare` . `locals` : `x = 1` , `sonuc = 1` .
- `cont` ile döngünün ikinci turunda yeniden durur: `sonuc = 4` .
- Kesme noktası kaldırılınca program biter: 1 + 4 + 9 = 14.

Sistematik Hata Ayıklama

1 **Yeniden üretmek**
Arızayı her seferinde ortaya çıkaran en küçük girdi bulunur (ör. `{-5, -2, -8}`).

Üretilemeyen hata ayıklanamaz.

Sistematik Hata Ayıklama

1 **Yeniden üretmek**
Arızayı her seferinde ortaya çıkaran en küçük girdi bulunur (ör. `{−5, −2, −8}`).

Üretilemeyen hata ayıklanamaz.

2 **Test olarak yazmak**
Bu girdiyle kalan bir birim testi yazılır: kırmızı test hatanın kanıtıdır.

Test, düzeltmenin işe yaradığını da gösterecek.

Sistematik Hata Ayıklama

1 **Yeniden üretmek**
Arızayı her seferinde ortaya çıkaran en küçük girdi bulunur (ör. `{-5, -2, -8}`).

Üretilemeyen hata ayıklanamaz.

2 **Test olarak yazmak**
Bu girdiyle kalan bir birim testi yazılır: kırmızı test hatanın kanıtıdır.

Test, düzeltmenin işe yaradığını da gösterecek.

3 **Daraltmak**
Yığın izindeki ilk satırdan başlanır; kesme noktasında değişkenler izlenir, sorunlu bölge ikiye bölünerek küçültülür.

Değerin ilk kez yanlış olduğu satır aranır. Koda yazılmış `assert` varsayımları (`-ea` ile açık) sorunu çıktığı yerde yakalar.

Sistematik Hata Ayıklama

1. **Yeniden üretmek**
   Arızayı her seferinde ortaya çıkaran en küçük girdi bulunur (ör. `{-5, -2, -8}`).

Üretilemeyen hata ayıklanamaz.

2. **Test olarak yazmak**
   Bu girdiyle kalan bir birim testi yazılır: kırmızı test hatanın kanıtıdır.

Test, düzeltmenin işe yaradığını da gösterecek.

3. **Daraltmak**
   Yığın izindeki ilk satırdan başlanır; kesme noktasında değişkenler izlenir, sorunlu bölge ikiye bölünerek küçültülür.

Değerin ilk kez yanlış olduğu satır aranır. Koda yazılmış `assert` varsayımları (`-ea` ile açık) sorunu çıktığı yerde yakalar.

4. **Hipotez kurmak ve sınamak**
   "`enb` 0 ile başladığı için güncellenmiyor" gibi bir açıklama önerilir; hata ayıklayıcıda doğrulanır.

Tahminle kod değiştirmek yeni hatalar doğurur.

Sistematik Hata Ayıklama

1. **Yeniden üretmek**
   Arızayı her seferinde ortaya çıkaran en küçük girdi bulunur (ör. `{-5, -2, -8}`).

   Üretilemeyen hata ayıklanamaz.

2. **Test olarak yazmak**
   Bu girdiyle kalan bir birim testi yazılır: kırmızı test hatanın kanıtıdır.

   Test, düzeltmenin işe yaradığını da gösterecek.

3. **Daraltmak**
   Yığın izindeki ilk satırdan başlanır; kesme noktasında değişkenler izlenir, sorunlu bölge ikiye bölünerek küçültülür.

   Değerin ilk kez yanlış olduğu satır aranır. Koda yazılmış `assert` varsayımları (`-ea` ile açık) sorunu çıktığı yerde yakalar.

4. **Hipotez kurmak ve sınamak**
   "`enb` 0 ile başladığı için güncellenmiyor" gibi bir açıklama önerilir; hata ayıklayıcıda doğrulanır.

   Tahminle kod değiştirmek yeni hatalar doğurur.

5. **Düzeltmek, tüm testleri çalıştırmak**
   Yeni test yeşile dönmeli, eskiler yeşil kalmalı (regresyon yok).

   Test kalıcı olarak test kümesinde kalır.

ANALİZ ARAÇLARI

Profil Oluşturucu (Profiler)

- Bir programın **zamanını nerede harcadığını** gösterir: hangi metot kaç kez çağrıldı, ne kadar sürdü, ne kadar bellek ayırdı.
- Profil oluşturucu **çağrı grafiğini** (call graph: kim kimi çağırdı) çeşitli biçimlerde gösterebilir: düz liste, ağaç, alev grafiği (flame graph).
- Teknik ayrıntılar, kullanılan derleyiciye ve JVM sürümüne göre **farklılık gösterir**.
- Kural: **önce ölç, sonra iyileştir**. Programın çoğu zaman küçük bir "sıcak nokta"da (hot spot) harcanır; tahminle optimizasyon çoğu kez yanlış yere yapılır.

Eski yöntem: `java -Xprof`

Eski JVM'lerin yerleşik profil oluşturucusuydu; **Java 10'da kaldırıldı**. Java 21'de:

```
OpenJDK 64-Bit Server VM warning: Ignoring
option -Xprof; support was removed in 10.0
```

Bugün

JDK Flight Recorder (JFR) + **JDK Mission Control**, **VisualVM**, IDE profil oluşturucuları. Aşağıda program JFR kaydıyla çalıştırılır, sonra kaydın özeti alınır.

```
java -XX:StartFlightRecording=filename=kayit.jfr Profil
jfr summary kayit.jfr
```

Sıcak Noktayı Ölçmek

```java
public class Profil {
  static String birlestirArti(int n) {
    String s = "";
    for (int i = 0; i < n; i++) {
      s += i;  // her turda yeni String
    }
    return s;
  }

  static String birlestirBuilder(int n) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < n; i++) {
      sb.append(i);
    }
    return sb.toString();
  }
  // main: yandaki kutuda
}
```

```java
public static void main(String[] args) {
  long t0 = System.nanoTime();
  String a = birlestirArti(50_000);
  long t1 = System.nanoTime();
  String b = birlestirBuilder(50_000);
  long t2 = System.nanoTime();
  long ms1 = (t1 - t0) / 1_000_000;
  long ms2 = (t2 - t1) / 1_000_000;
  System.out.println("+= : " + ms1 + " ms");
  System.out.println("SB : " + ms2 + " ms");
  System.out.println(a.equals(b));
}
```

- `s += i` her turda **yeni bir String** oluşturup eskisini kopyalar: toplam iş n^2 ile büyür.
- `StringBuilder` tek bir tampona ekler: iş n ile büyür. Fark yüzlerce kat.



Bir çalıştırmada çıktı (makineye göre değişir)

```
+= : 873 ms
SB : 2 ms
true
```

ANALİZ ARAÇLARI

Kod Kapsamı Aracı (Coverage Tool)

- Test çalıştırması sırasında kodun **hangi bölümlerinin çalıştırıldığını** gösterir.
- **Çalıştırılmayan** veya kullanılmayan kod parçaları belirlenebilir.
- **Test edilmemiş** kod bölgelerini bulmaya yardımcı olur.
- Kod kapsamı, yazılımın **güvenilirliğini** artırmak için kritik bir araçtır.
- Araçlar: **JaCoCo** (Maven / Gradle), IntelliJ "Run with Coverage", Eclipse **EclEmma**. Sonuç, kod satırları renklendirilerek gösterilir.

Ölçüt	Ne sayılır?
Satır (line)	En az bir kez çalışan satırlar
Dal (branch)	Her `if` / döngü koşulunun hem doğru hem yanlış sonucu
Metot	En az bir kez çağrılan metotlar

%100 kapsam ≠ hatasız kod. Kapsam yalnızca satırın **çalıştığını** söyler, sonucun **doğru** olduğunu değil: hiç `assert` içermeyen bir test de kapsamı artırır.

Kod Kapsamı: Bir Örnek

`Isaret` : İKİ TEST SONRASI KAPSAM

```
public class Isaret {
  public static String isaret(int x) {
    if (x > 0) {
      return "pozitif";
    } else if (x < 0) {
      return "negatif";
    }
    return "sıfır";
  }
}
```

çalıştı hiç çalışmadı

Ölçüt	Kapsam
Satır (çalıştırılabilir 5 satır)	4 / 5 = %80
Dal (2 koşul × doğru / yanlış)	3 / 4 = %75

`IsaretTest`

```
class IsaretTest {
  @Test
  void pozitif() {
    assertEquals("pozitif", Isaret.isaret(5));
  }

  @Test
  void negatif() {
    assertEquals("negatif", Isaret.isaret(-3));
  }
}
```

İki test de yeşil (✓), ama `x < 0` koşulunun **yanlış** dalı ve `return "sıfır"` hiç denenmedi. Kapsam raporu eksik testi gösterir: `isaret(0)`.

Kod Kapsamı Adım Adım

```
1  public class Isaret {
▶    public static String isaret(int x) {
3      if (x > 0) {
4        return "pozitif";
5      } else if (x < 0) {
6        return "negatif";
7      }
8      return "sıfır";
9    }
10 }
```

satırlar

3	4	5	6	8

0 / 5 = %0

dallar

x > 0 doğru	x > 0 yanlış
x < 0 doğru	x < 0 yanlış

0 / 4 = %0

testler: –

Çalıştırılabilir 5 satır (3, 4, 5, 6, 8) ve 2 koşulun 4 dalı var. Hiç test çalışmadı: hepsi kırmızı.

Adım 1 / 8

DEĞİŞKENLER

`satır = 0/5` `dal = 0/4`

ÇAĞRI YIĞINI

boş

Kod Kapsamı Adım Adım

```
1  public class Isaret {
2    public static String isaret(int x) {
▶      if (x > 0) {
4        return "pozitif";
5      } else if (x < 0) {
6        return "negatif";
7      }
8      return "sıfır";
9    }
10 }
```

satırlar

3	4	5	6	8

1 / 5 = %20

dallar

x > 0 doğru	x > 0 yanlış
x < 0 doğru	x < 0 yanlış

1 / 4 = %25

testler: pozitif(5)

`isaret(5)` : 3. satır çalışır, koşul **doğru** dalına gider.

Adım 2 / 8

DEĞİŞKENLER

`x = 5`

ÇAĞRI YIĞINI

`isaret(5)`

`pozitif()`

Kod Kapsamı Adım Adım

```
1  public class Isaret {
2    public static String isaret(int x) {
3      if (x > 0) {
▶        return "pozitif";
5      } else if (x < 0) {
6        return "negatif";
7      }
8      return "sıfır";
9    }
10 }
```

satırlar

3	4	5	6	8	2 / 5 = %40

dallar

x > 0 doğru	x > 0 yanlış	1 / 4 = %25
x < 0 doğru	x < 0 yanlış	

testler: pozitif(5)

"pozitif" döner; test geçer. Kapsam: 2/5 satır, 1/4 dal.

Adım 3 / 8

DEĞİŞKENLER

x = **5**

ÇAĞRI YIĞINI

isaret(5)

pozitif()

Kod Kapsamı Adım Adım

```
1  public class Isaret {
2    public static String isaret(int x) {
3      if (x > 0) {
4        return "pozitif";
▶      } else if (x < 0) {
6        return "negatif";
7      }
8      return "sıfır";
9    }
10 }
```

satırlar

3 | 4 | 5 | 6 | 8 — 3 / 5 = %60

dallar

x > 0 doğru | x > 0 yanlış | x < 0 doğru | x < 0 yanlış — 3 / 4 = %75

testler: pozitif(5), negatif(-3)

`isaret(-3)` : `x > 0` yanlış, `x < 0` doğru. İki yeni dal.

Adım 4 / 8

DEĞİŞKENLER

`x = -3`

ÇAĞRI YIĞINI

`isaret(-3)`

`negatif()`

Kod Kapsamı Adım Adım

```
1  public class Isaret {
2    public static String isaret(int x) {
3      if (x > 0) {
4        return "pozitif";
5      } else if (x < 0) {
▶        return "negatif";
7      }
8      return "sıfır";
9    }
10 }
```

satırlar

3	4	5	6	8

4 / 5 = %80

dallar

x > 0 doğru	x > 0 yanlış
x < 0 doğru	x < 0 yanlış

3 / 4 = %75

testler: pozitif(5), negatif(-3)

"negatif" döner. İki test de yeşil.

Adım 5 / 8

DEĞİŞKENLER

`x = -3`

ÇAĞRI YIĞINI

`isaret(-3)`

`negatif()`

Kod Kapsamı Adım Adım

```
1  public class Isaret {
2    public static String isaret(int x) {
3      if (x > 0) {
4        return "pozitif";
5      } else if (x < 0) {
6        return "negatif";
7      }
▶      return "sıfır";
9    }
10 }
```

satırlar



4 / 5 = %80

dallar



3 / 4 = %75

testler: pozitif(5), negatif(-3)

Rapor: satır %80, dal %75. 8. satır ve `x < 0` 'ın yanlış dalı **hiç denenmedi**: eksik test `isaret(0)`.

Adım 6 / 8

DEĞİŞKENLER

`satır = 4/5` `dal = 3/4`

ÇAĞRI YIĞINI

boş

Kod Kapsamı Adım Adım

```
1  public class Isaret {
2    public static String isaret(int x) {
▶      if (x > 0) {
4        return "pozitif";
▶      } else if (x < 0) {
6        return "negatif";
7      }
8      return "sıfır";
9    }
10 }
```

satırlar

3	4	5	6	8

4 / 5 = %80

dallar

x > 0 doğru	x > 0 yanlış
x < 0 doğru	x < 0 yanlış

4 / 4 = %100

testler: pozitif(5), negatif(-3), sifir(0)

Yeni test `isaret(0)` : iki koşul da yanlış. Dal kapsamı 4/4.

Adım 7 / 8

DEĞİŞKENLER

`x = 0`

ÇAĞRI YIĞINI

`isaret(0)`

`sifir()`

Kod Kapsamı Adım Adım

```
1  public class Isaret {
2    public static String isaret(int x) {
3      if (x > 0) {
4        return "pozitif";
5      } else if (x < 0) {
6        return "negatif";
7      }
▶      return "sıfır";
9    }
10 }
```

satırlar

3	4	5	6	8

5 / 5 = %100

dallar

x > 0 doğru	x > 0 yanlış
x < 0 doğru	x < 0 yanlış

4 / 4 = %100

testler: pozitif(5), negatif(-3), sifir(0)

"sıfır" döner: satır ve dal kapsamı %100. Yine de kapsam yalnızca satırların **çalıştığını** söyler; doğruluğu `assertEquals` sınar.

Adım 8 / 8

DEĞİŞKENLER

`x = 0`

ÇAĞRI YIĞINI

`isaret(0)`

`sifir()`

ANALİZ ARAÇLARI

Statik Analiz: FindBugs ve SpotBugs

- **FindBugs**, Java kodunun **statik analizi** için kullanılan bir araçtır: programı **çalıştırmadan** derlenmiş kodu inceler.
- Kod hatalarını **hızlı bir şekilde** bulmaya yardımcı olur: `null` erişimi, yanlış karşılaştırma, kapatılmayan kaynak…
- FindBugs artık geliştirilmiyor; devamı **SpotBugs**'tır (Maven, Gradle, IDE eklentileri). Benzerleri: PMD, Checkstyle, SonarQube.

Girdi `evet` olduğu hâlde çıktı: `Reddedildi`. Derleyici sessiz kalır; SpotBugs iki satırı da işaretler.

```java
import java.util.Scanner;

public class SpotOrnek {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String cevap = sc.nextLine();
    // sonuç kullanılmıyor:
    cevap.trim();
    // String == ile karşılaştırılıyor:
    if (cevap == "evet") {
      System.out.println("Onaylandı");
    } else {
      System.out.println("Reddedildi");
    }
  }
}
```

SpotBugs kalıbı	Uyarı
`ES_COMPARING_STRINGS_WITH_EQ`	String nesneleri `==` ile karşılaştırılıyor; doğrusu `equals`
`RV_RETURN_VALUE_IGNORED`	`trim()` yeni bir String döndürür; sonuç atılıyor

İlk Statik Analizci: Derleyici (`-Xlint`)

```java
public class Lint {
  public static void main(String[] args) {
    int gun = 2;
    switch (gun) {
      case 1:
        System.out.println("Pazartesi");
      case 2:
        System.out.println("Salı");
      case 3:
        System.out.println("Çarşamba");
        break;
    }
    int x = 10 / 0;
  }
}
```

java Lint

```
Salı
Çarşamba
Exception in thread "main"
java.lang.ArithmeticException: / by zero
    at Lint.main(Lint.java:13)
```

javac -Xlint:all Lint.java

```
Lint.java:13: warning: [divzero] division by
zero
    int x = 10 / 0;
                ^
Lint.java:7: warning: [fallthrough] possible
fall-through into case
      case 2:
      ^
Lint.java:9: warning: [fallthrough] possible
fall-through into case
      case 3:
      ^
3 warnings
```

- Düz `javac Lint.java` **hiç uyarı vermeden** derler.
- `-Xlint:all` şüpheli yapıları bildirir: `break` unutulmuş `case`'ler ve sabit sıfıra bölme.
- Uyarıları okumak, hatayı **çalıştırmadan önce** bulmanın en ucuz yoludur.

QuickCheck: Özellik Tabanlı Test

- **QuickCheck** testleri **otomatik olarak üretir**: girdileri programcı değil, araç rastgele seçer.
- Programcı tek tek örnek yerine bir **özellik** (property) yazar: "her girdi için doğru olmalı".
- Özelliği bozan bir girdi bulunursa onu **karşı örnek** olarak raporlar (ve mümkünse küçültür).
- İlki Haskell içindir; Java dahil **birçok dil için** sürümleri vardır: **jqwik**, **junit-quickcheck**.

Yandaki test fikri düz JUnit ile uygular: 1000 rastgele metin.

Test Sonuçları

✓ `ikiKezTers…()`

1 test (1000 girdi) · başarılı

```
class OzellikTest {

  static String rastgeleMetin(Random r) {
    int uzunluk = r.nextInt(20);  // 0..19
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < uzunluk; i++) {
      sb.append((char) ('a' + r.nextInt(26)));
    }
    return sb.toString();
  }

  @Test
  void ikiKezTersCevirmekAyniMetniVerir() {
    Random r = new Random(42);    // sabit tohum
    for (int k = 0; k < 1000; k++) {
      String s = rastgeleMetin(r);
      String t = MetinIslem.ters(s);
      assertEquals(s, MetinIslem.ters(t));
    }
  }
}
```

Özellik Testi İlk Sürümdeki Hatayı Buluyor

```
class OzellikTest2 {

  @Test
  void enBuyukDizininEnBuyukElemanidir() {
    Random r = new Random(7);
    for (int k = 0; k < 1000; k++) {
      int[] dizi = new int[1 + r.nextInt(5)];
      for (int i = 0; i < dizi.length; i++) {
        dizi[i] = r.nextInt(201) - 100;     // -100..100
      }
      int enb = DiziIslem.enBuyuk(dizi);
      boolean dizideVar = false;
      for (int x : dizi) {
        assertTrue(enb >= x);               // özellik 1
        if (x == enb) dizideVar = true;
      }
      assertTrue(dizideVar,                   // özellik 2
        "Karşı örnek: " + Arrays.toString(dizi));
    }
  }
}
```

İlk (hatalı) `enBuyuk` sürümüyle çalıştırıldığında:

Test Sonuçları

✗ `enBuyukDizininEnBuyukElemanidir()`
Karşı örnek: [-60, -69, -39, -65, -13] ==> expected: <true> but was: <false>

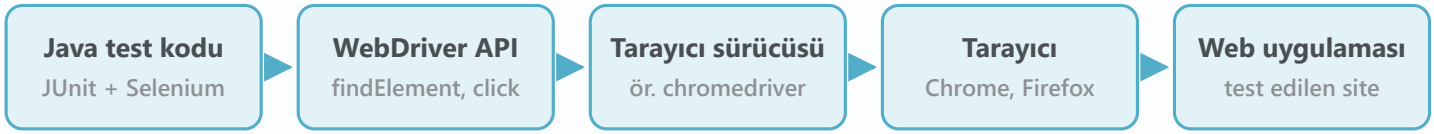
1 test · 1 başarısız

- **Özellik 1** (sonuç hiçbir öğeden küçük değil) hatalı sürümde de **geçer**: 0, tüm negatiflerden büyüktür.
- **Özellik 2** (sonuç dizinin bir öğesidir) hatayı yakalar.
- Düzeltilmiş sürümle 1000 girdinin tamamı geçer. Ders: özellikler **eksiksiz** yazılmalıdır; zayıf bir özellik hatalı kodu da onaylar.

ANALİZ ARAÇLARI

Selenium

- **Selenium**, web uygulamalarını test etmek için bir **çerçevedir** (framework).
- **Tarayıcı tabanlı** testleri kolaylaştırır: gerçek bir tarayıcıyı programla açar, bağlantıya tıklar, forma yazar, butona basar ve sayfada beklenen metnin çıktığını doğrular.
- Java, Python, C#, JavaScript gibi dillerle kullanılır; Java'da genellikle JUnit testinin içinden çağrılır.



komutlar soldan sağa gider; sayfanın durumu (metin, başlık) teste geri döner

Tipik senaryo
Giriş sayfası açılır → kullanıcı adı ve parola yazılır → "Giriş" → başlıkta "Hoş geldiniz" var mı?

Test düzeyi
Kullanıcının gördüğü bütünü sınar: **sistem** ve **kabul** testleri.

Maliyet
Birim testlerinden çok daha yavaş ve kırılgandır; sayısı az tutulur (piramidin tepesi).

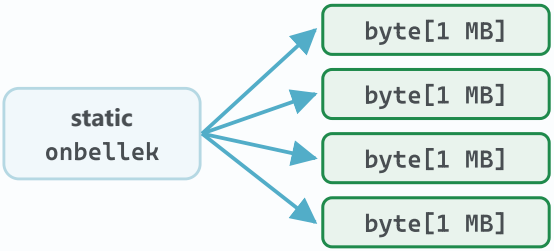
Bellek Sızıntısı Örneği

```java
import java.util.ArrayList;
import java.util.List;

public class Sizinti {
  static List<byte[]> onbellek = new ArrayList<>();

  static void istekIsle(int no) {
    byte[] veri = new byte[1024 * 1024];  // 1 MB
    onbellek.add(veri);                   // hiç silinmiyor!
  }

  public static void main(String[] args) {
    for (int i = 1; ; i++) {
      istekIsle(i);
      if (i % 20 == 0) {
        System.out.println(i + " istek işlendi");
      }
    }
  }
}
```



hepsine hâlâ erişilebiliyor → GC silemez

java -Xmx256m Sizinti

```
20 istek işlendi
…
120 istek işlendi
Exception in thread "main"
java.lang.OutOfMemoryError: Java heap
space
    at
Sizinti.istekIsle(Sizinti.java:8)
    at Sizinti.main(Sizinti.java:14)
```

Çöp toplayıcı (GC) yalnız **erişilemeyen** nesneleri siler; sızıntı, gereksiz nesnelere **hâlâ bir referans** kalmasıdır.
`-Xmx256m` : heap en çok 256 MB.

ANALİZ ARAÇLARI

Eclipse Memory Analyzer (MAT)

- Java **heap belleğini analiz ederek** bellek sızıntılarını tespit eder.
- **Bellek optimizasyonu** için kullanılır: hangi nesne ne kadar yer tutuyor, onları kim tutuyor?
- Girdi: bir **heap dökümü** (heap dump, `.hprof`): belirli bir andaki tüm nesnelerin fotoğrafı.
- Önemli raporlar: **Leak Suspects** (sızıntı şüphelileri), **Dominator Tree** (en çok belleği kim tutuyor), **Histogram** (sınıf başına nesne sayısı).

HEAP DÖKÜMÜ ALMAK

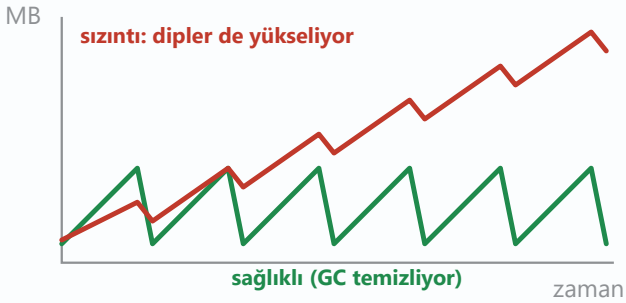
```
// bellek dolunca otomatik
java -XX:+HeapDumpOnOutOfMemoryError Sizinti
// çalışan programdan
jps
jcmd <pid> GC.heap_dump yigin.hprof
```

Örnekte ne görülür?

Leak Suspects raporu, heap'in neredeyse tamamını `Sizinti.onbellek` listesinin ve içindeki `byte[]` dizilerinin tuttuğunu gösterir. Çözüm: işi biten veriyi listeden çıkarmak ya da boyutu sınırlı bir önbellek kullanmak.

VisualVM

- **Bellek tüketimini**, **iş parçacığı kullanımını**, CPU yükünü ve daha fazlasını **canlı** olarak görselleştirir.
- Java uygulamalarının **performansını izlemek ve analiz etmek** için kullanılır: çalışan JVM'lere bağlanır, profil çıkarır, heap ve iş parçacığı dökümü alır.
- Ücretsiz ve açık kaynaktır (visualvm.github.io); eskiden JDK ile birlikte `jvisualvm` adıyla gelirdi.
- Komut satırı karşılıkları: `jps` (JVM'leri listele), `jstat`, `jcmd`, `jconsole`.



Yığın grafiği bir sızıntıyı ele verir: çöp toplamadan sonraki dipler sürekli yükseliyorsa nesneler serbest bırakılmıyordur. Sonra heap dökümü alınıp MAT ile incelenir.

ÖZET

Özet: Hangi Soruya Hangi Araç?

Soru	Teknik / araç
Bu metot doğru sonuç veriyor mu?	**Birim testi**: JUnit, `@Test`, `assertEquals`, `assertThrows`
Parçalar birlikte, bütün sistem ve kullanıcı gözünden çalışıyor mu?	Modül, alt sistem, sistem, **kabul** (alfa / beta) testi; web için **Selenium**
Değişiklik eski bir şeyi bozdu mu?	**Regresyon testi**: tüm otomatik testler her değişiklikte (CI) çalıştırılır
Hangi girdileri seçmeliyim?	**Sınır değerleri**, denklik sınıfları, `@ParameterizedTest`
Seçmediğim girdilerde de doğru mu?	**Özellik tabanlı test** (QuickCheck, jqwik)
Testlerim kodun ne kadarını çalıştırıyor?	**Kod kapsamı**: JaCoCo, EclEmma
Kodda çalıştırmadan görülebilen hata var mı?	**Statik analiz**: `javac -Xlint`, SpotBugs (FindBugs)
Neden yanlış çalışıyor?	**Hata ayıklayıcı**: kesme noktası, Step Into / Over / Out, izleme; `jdb`
Zaman nerede harcanıyor?	**Profil oluşturucu**: JFR, VisualVM
Bellek neden doluyor?	**VisualVM** ile izleme, heap dökümünü **Eclipse Memory Analyzer** ile inceleme

ÖZET

İlgili Sayfalar

Çalışma Soruları

Birim testi, sınır değerleri, kod kapsamı ve hata ayıklama üzerine çözümlü sorular.

Terim sözlüğü

Birim testi, doğrulama, regresyon testi, kod kapsamı, hata ayıklayıcı gibi terimlerin Türkçe ve İngilizce karşılıkları.

Test edilebilir kod genellikle iyi tasarlanmış koddur. Bağımlılıkları dışarıdan almak (`Zar` örneğindeki `Random` gibi) hem testi hem de tasarımı kolaylaştırır.

ÖZET

Sık Yapılan Hatalar

Beklenen ile gerçeği ters yazmak

`assertEquals(hm.topla(2, 3), 5)` da çalışır ama kalınca mesaj ters okunur: `expected` gerçek sonucu gösterir.

Ondalıkta delta unutmak

`assertEquals(0.3, 0.1 + 0.2)` kalır: `expected: <0.3> but was: <0.30000000000000004>`.

`import static` eksik

`assertEquals(5, x)` → derleme hatası: `cannot find symbol: method assertEquals(int,int)`.

JUnit 4 ile 5'i karıştırmak

`org.junit.Test` ile `org.junit.jupiter.api.Test` farklıdır; yanlış `import` testlerin hiç çalışmamasına yol açabilir.

`@BeforeAll` metodu `static` değil

JUnit 5 testleri çalıştırmaz: `@BeforeAll method '…' must be static unless the test class is annotated with @TestInstance(Lifecycle.PER_CLASS).`

Testler arası bağımlılık

Bir testin bıraktığı nesneye ötekinde güvenmek: sıra değişince testler rastgele kalır. Her test kendi verisini kurmalıdır.

Doğrulaması olmayan test

Yalnızca metodu çağırıp hiçbir `assert` yazmamak: test her zaman ✓ olur, hiçbir şey kanıtlamaz.

Yalnız "mutlu yol"u test etmek

Sınırlar (0, boş dizi, negatif), geçersiz girdiler ve istisnalar unutulur; hatalar tam oradadır.

ÖZET

Örnek Problemler (1)

PROBLEM 1: HANGİ TEST GEÇER, HANGİSİ KALIR?

```
class SoruTest {
  @Test void a() { assertEquals(7, 3 + 4); }
  @Test void b() { assertEquals(0.3, 0.1 + 0.2); }
  @Test void c() { assertTrue("Java".length() == 4); }
  @Test void d() { assertThrows(ArithmeticException.class,
      () -> System.out.println(10 / 2)); }
}
```

PROBLEM 2

`harfNotu` içinde `puan >= 50` yanlışlıkla `puan > 50` yazılsaydı parametreli testin hangi satırı kalırdı? Mesaj ne olurdu?

PROBLEM 3

Aşağıdaki fonksiyon için en az kaç test gerekir ki **dal kapsamı** %100 olsun? Girdileri seçin.

```
public class Mutlak {
  static int mutlak(int x) {
    if (x < 0) {
      return -x;
    }
    return x;
  }
}
```

▶ **Cevapları göster**

ÖZET

Örnek Problemler (2)

PROBLEM 4: BU TEST SINIFININ SORUNU NEDİR?

```
class SayacTest {
  static int sayac = 0;

  @Test void artir() {
    sayac++;
    assertEquals(1, sayac);
  }

  @Test void ikiKezArtir() {
    sayac += 2;
    assertEquals(2, sayac);
  }
}
```

PROBLEM 5

`AdimOrnek` programında hata ayıklayıcı `int s = kareToplam(3, 4);` satırında durmuşken art arda iki kez **Step Over** yapılırsa hangi satıra gelinir, ekranda ne vardır ve `s` kaçtır?

▶ **Cevapları göster**

ÖZET

Örnek Problemler (3): Hangi Araç?

Problem 6: Her durum için hangi teknik ya da araç kullanılmalı?

1. Program 10 dakika sonra `OutOfMemoryError` ile çöküyor.
2. Rapor oluşturma 30 saniye sürüyor; hangi metot yavaş?
3. Yeni eklenen indirim kodu eski fatura hesabını bozdu mu?
4. Web sitesindeki kayıt formu tüm tarayıcılarda çalışıyor mu?
5. `switch` 'te unutulmuş `break` 'ler var mı?
6. Bir döngü 10 000. turda yanlış değer üretiyor; o anda değişkenler ne?

▶ **Cevapları göster**

ÖZET

Alıştırmalar (1)

1. `Dikdortgen` sınıfı (`en` , `boy` , `alan()` , `cevre()`) için JUnit 5 test sınıfı yazın. Negatif kenar uzunluğu `IllegalArgumentException` fırlatmalı.
2. `static boolean asalMi(int n)` için `@ParameterizedTest` ile sınır değerlerini (0, 1, 2, 3, 4, 97, 100) test edin.
3. `BankaHesabi` 'na `havale(BankaHesabi hedef, double miktar)` ekleyin; önce testini yazın (TDD). Hangi testler gerekir?

▶ **Cevapları göster**

ÖZET

Alıştırmalar (2)

4. Bir dizinin ortalamasını hesaplayan metot bazı dizilerde yanlış sonuç veriyor (ör. `{1, 2}` için 1.0). Hata ayıklayıcıyla kusuru bulma adımlarını yazın.
5. "Sıralanmış dizide her öğe bir sonrakinden büyük değildir" özelliğini 1000 rastgele dizi ile sınayan bir test yazın. Bu özellik tek başına yeterli mi?
6. `Isaret` sınıfının dal kapsamını %100 yapmak için hangi test eklenmeli?

▶ **Cevapları göster**

ÖZET

Kaynaklar

- JUnit Team, *JUnit 5 User Guide*, junit.org/junit5/docs/current/user-guide (Writing Tests, Assertions, Parameterized Tests).
- Oracle, *The Java Tutorials* ve *JDK 21 Tool Specifications*: `javac` (`-Xlint`), `jdb`, `jcmd`, `jfr`.
- Oracle, *Programming With Assertions* (Java SE belgeleri).
- J. Gosling ve diğ., *The Java Language Specification*, Java SE 21 Edition (14.10 The assert Statement).
- P. Deitel ve H. Deitel, *Java: How to Program*, 11. baskı, Pearson.
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12. baskı, Pearson (Bölüm 7: Exceptions, Assertions, and Logging; hata ayıklama ipuçları).
- K. Beck, *Test-Driven Development: By Example*, Addison-Wesley.
- Araç belgeleri: SpotBugs (spotbugs.github.io), JaCoCo (jacoco.org), Selenium (selenium.dev), Eclipse Memory Analyzer (eclipse.dev/mat), VisualVM (visualvm.github.io), jqwik (jqwik.net).

Son

Bölüm 7: Test ve Hata Ayıklama

Sonraki bölüm: Bölüm 8: Giriş Çıkış İşlemleri · Terim sözlüğü