MTK467 Nesneye Yönelik Programlama

Hafta 9 - Sınıflar ve Nesneler

Zümra Kavafoğlu
https://zumrakavafoglu.github.io/

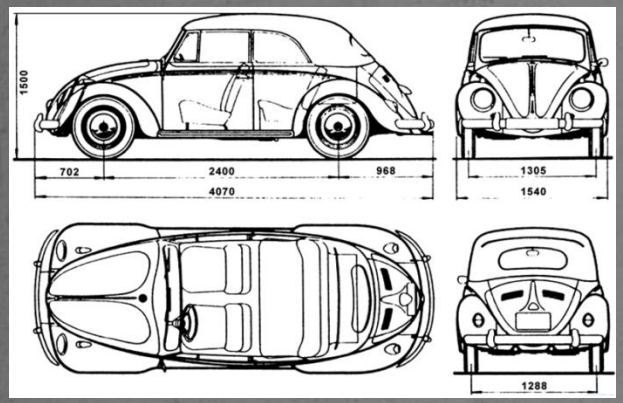
SINIF

- En genel haliyle sınıf kullanıcının tanımladığı bir veri tipidir. Nesne ise o tipte oluşturulmuş bir değişken gibi düşünülebilir.

- Java Sınıfı = veri + metotlar

NESNE

SINIF

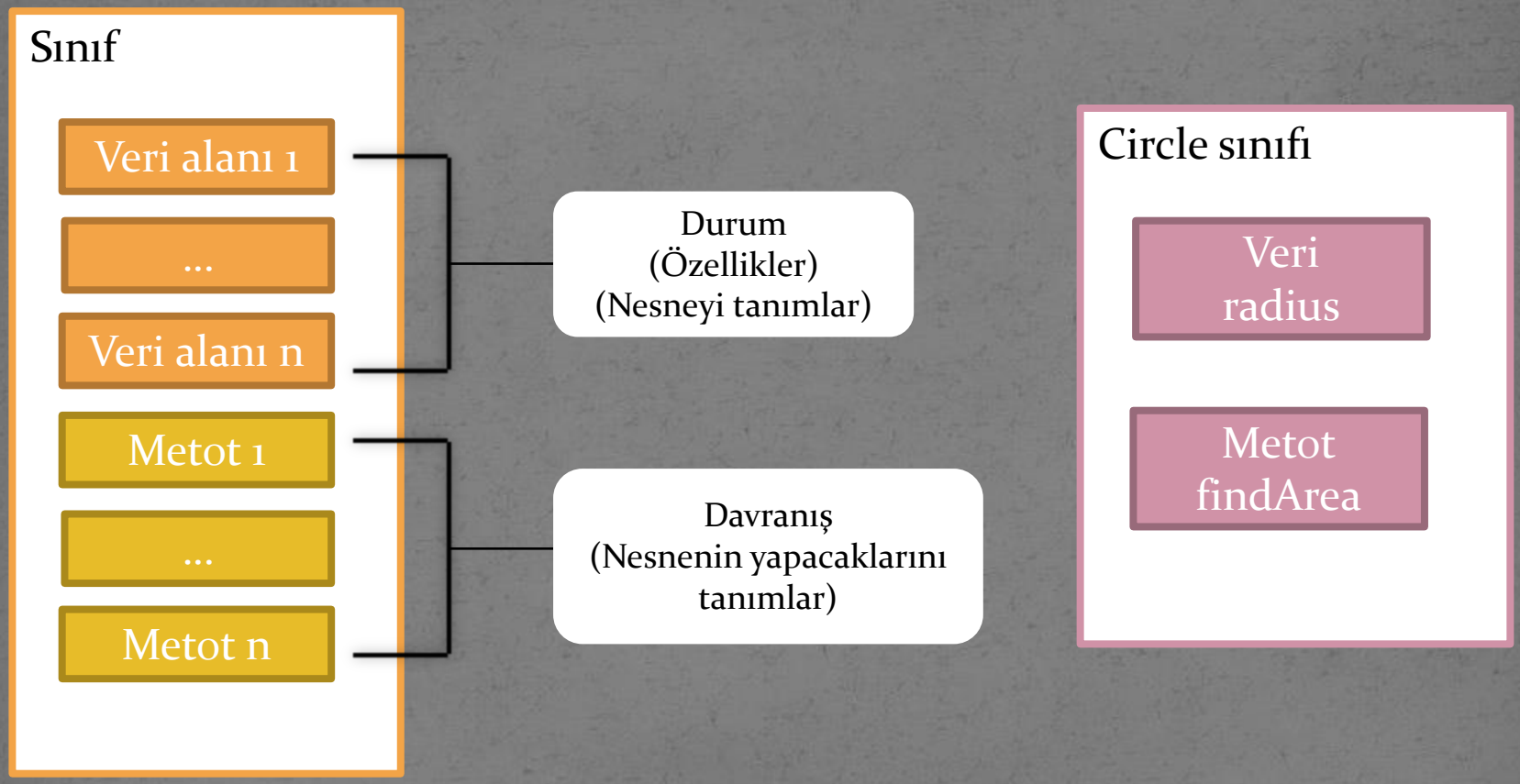


JAVA

NESNE

Sınıflarla nesneler arasındaki ilişki otomobil tasarımıyla otomobil arasındaki ilişki gibidir.
Tasarımda tanımlananlarla otomobilleri yaparsınız.
Ama tasarımı değil otomobili sürebilirsiniz.

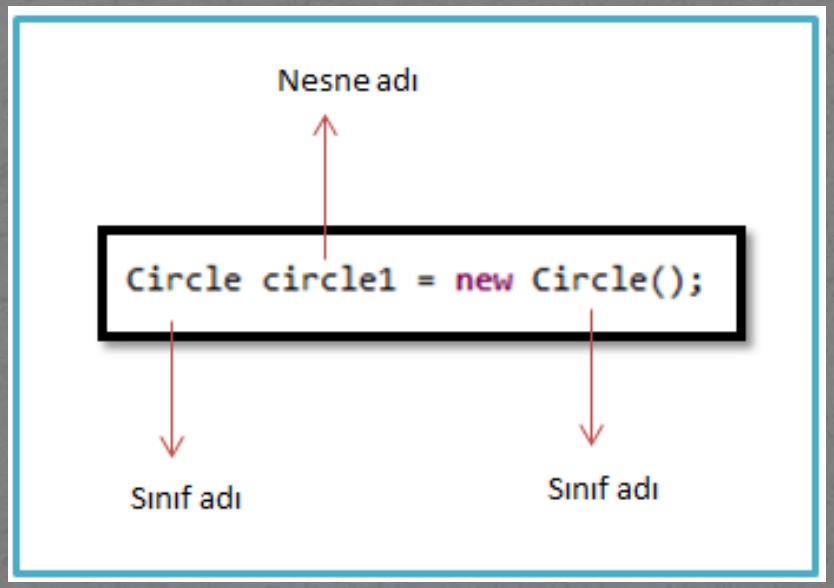
SINIF


Sınıf
Veri alanı 1
...
Veri alanı n
Metot 1
...
Metot n
Durum
(Özellikler)
(Nesneyi tanımlar)
Davranış
(Nesnenin yapacaklarını
tanımlar)
Circle sınıfı
Veri
radius
Metot
findArea

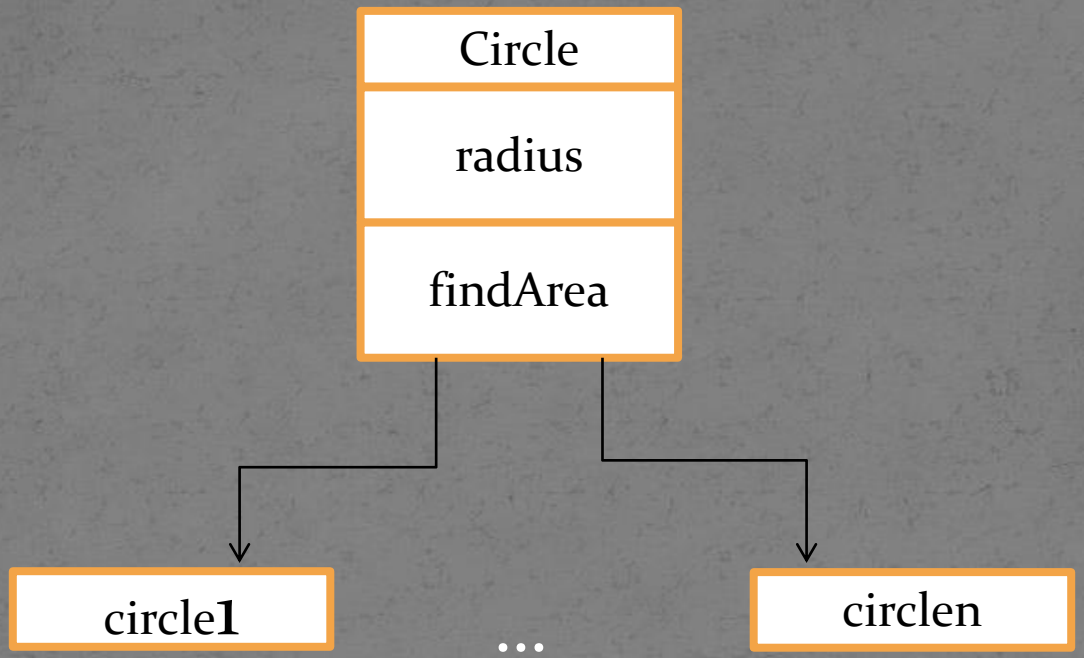
```
class Circle {

    double radius = 1.0;

    double findArea()
    {
        return radius*radius*Math.PI;
    }
}
```

- Bu sınıf şimdiye kadar gördüğümüz sınıflardan farklıdır: **Bu sınıfta main yoktur.** Dolayısıyla bu sınıfı çalıştıramazsınız.
- **Bu sınıf yalnızca Circle nesnelerini açıklamak ve üretmek için kullanılan tanımlamalardır. Yani sınıf bir nesnenin verilerinin ve metotlarının ne olacağını tanımlar.**

Nesne oluşturma

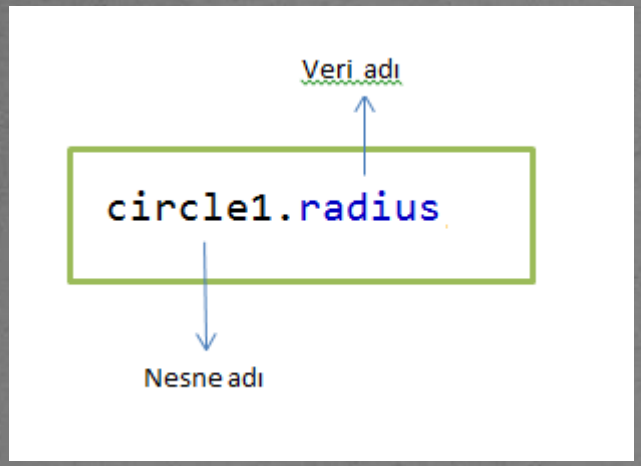


NESNE

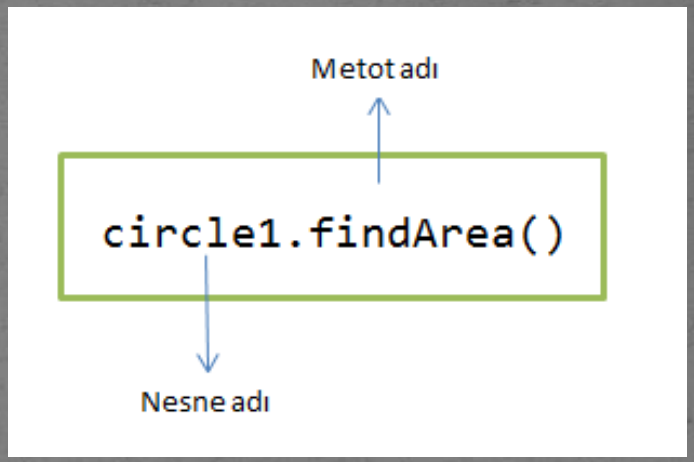


Nesnenin veri ve metotlarına ulaşma

Veriye ulaşma



Metoda ulaşma



YAPILANDIRICILAR (CONSTRUCTORS)

- Tanımladığımız Circle sınıfından üreteceğimiz her nesne aynı yarıçapa(1.0) sahip olacaktır. Ancak aynı Circle sınıfından yarıçapları farklı nesneler üretebilmek daha kullanışlı değil midir?
- Java, sınıflarında, sınıfın verilerine ilk değer vermek için kullanılabilen, *yapılandırıcı* olarak adlandırılan, özel bir metot tanımlamaya izin verir.
- Yapılandırıcının ismi sınıfınkiyle aynı olmalıdır.

YAPILANDIRICILAR (CONSTRUCTORS)

Circle sınıfının, radius verisine ilk değer veren yapılandırıcısının tanımlanması

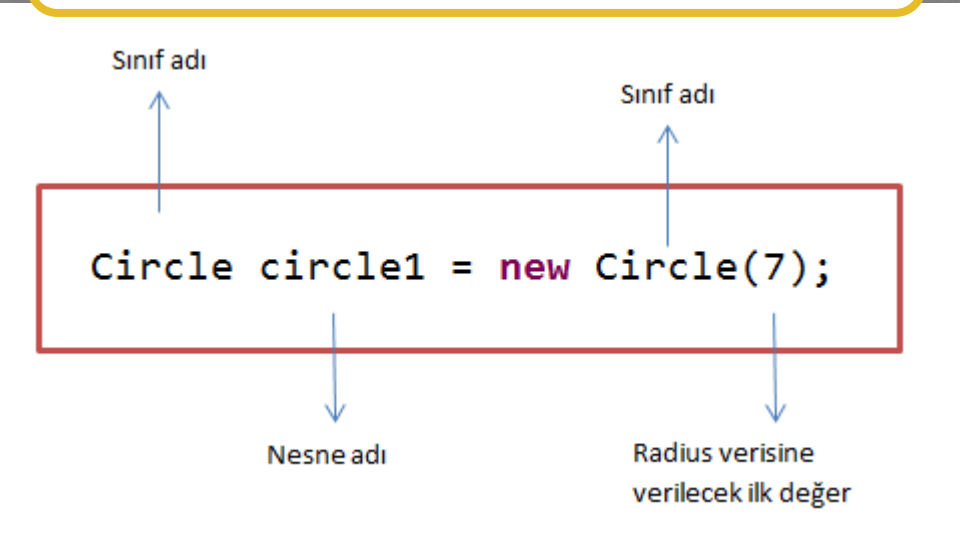
```
class Circle {

    double radius;

    Circle(double r)
    {
        radius = r;
    }

    double findArea()
    {
        return radius*radius*Math.PI;
    }
}
```

Sınıf adı

Yapılandırıcı

YAPILANDIRICI İLE NESNE OLUŞTURMA

radius verisinin değeri 7 olan, circle1 isimli Circle nesnesinin oluşturulması



YAPILANDIRICILAR (CONSTRUCTORS)

- Yapılandırıcılar iki yönden diğer metotlardan farklıdırlar:
  - Yapılandırıcıların dönüş tipi yoktur.
  - Yapılandırıcılar başlarına *new* koyularak çağırılırlar.

- **radius verisine başka bir metotla da değer verilemez miydi? Ya da değerini sonradan main'in içinde belirleyemez miydim? Fark ne?**
- **Fark yapılandırıcının nesneyi radius değerini belirleyerek oluşturması. Herhangi başka bir metot kullanılarak yapılırsa, önce nesne başka bir radius değeriyle oluşturulur, sonra metot çağırıldığında radius istediğimiz değer yapılır.**

VARSAYILAN YAPILANDIRICI (DEFAULT CONSTRUCTOR)

- Sınıftaki verilere default (varsayılan) bir ilk değer vermek için Varsayılan Yapılandırıcılar oluşturulur.

- Varsayılan Yapılandırıcılar parametre almaz, onun dışında bildiğimiz yapılandırıcılarla aynı biçimde tanımlanır ve kullanılırlar.

VARSAYILAN YAPILANDIRICININ TANIMLANMASI

Circle sınıfının, radius verisine default ilk değerini veren varsayılan yapılandırıcısının tanımlanması

```
class Circle {

    double radius;

    Circle()
    {
        radius = 1.0;
    }

    Circle(double r)
    {
        radius = r;
    }

    double findArea()
    {
        return radius*radius*Math.PI;
    }
}
```

Sınıf adı

Varsayılan Yapılandırıcı

Radius verisine ilk değerin verilmesi

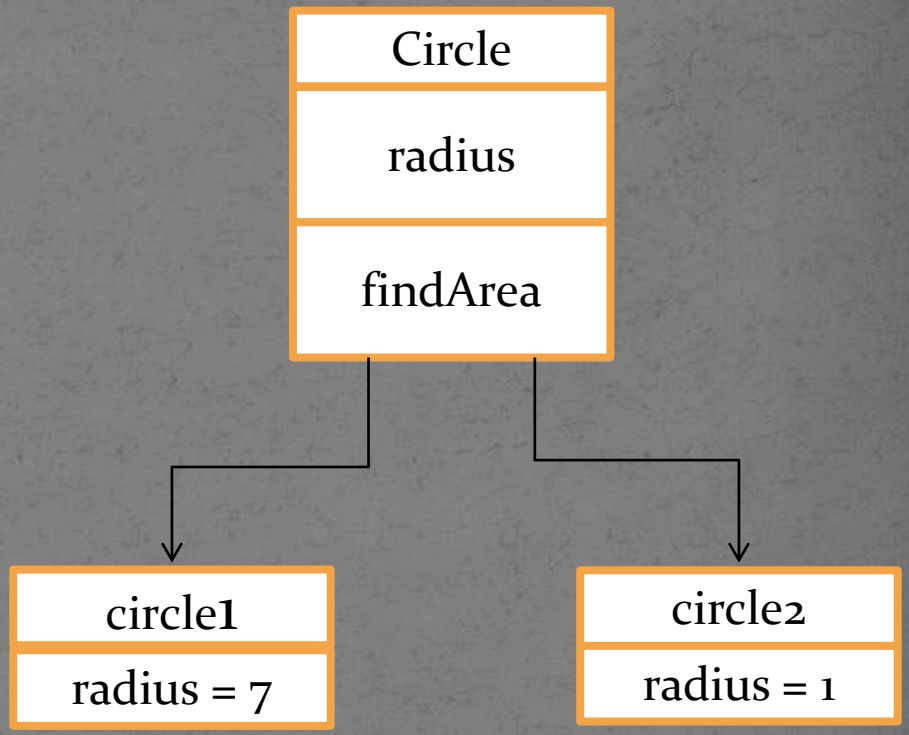
VARSAYILAN YAPILANDIRICIYLA NESNE OLUŞTURMA

radius verisinin değeri 1 olan circle2 isimli Circle nesnesinin oluşturulması

```
Circle circle2 = new Circle();
```

YAPILANDIRICIYLA NESNE OLUŞTURMA

```
Circle circle1 = new Circle(7);

Circle circle2 = new Circle();
```



UYARI:

1. Eğer bir sınıfta hiç yapılandırıcı(varsayılan ya da değil) tanımlanmadıysa, sınıfın bir nesnesi new SınıfAdı() komutuyla oluşturulabilir. Ancak eğer sınıfta sadece varsayılan olmayan yapılandırıcı tanımlandıysa sınıfın new SınıfAdı() komutuyla bir nesnesi oluşturulamaz.
2. Bir sınıf verisi, nümerik bir tipteyse 0, boolean tipteyse false,char tipindeyse '\u0000' ve bir nesneyse null ilk değerini alır.

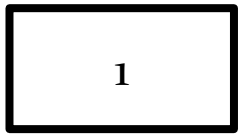
- Birlikte bir Rectangle sınıfı oluşturalım…

Nesnelerin metotlara parametre olarak verilmesi

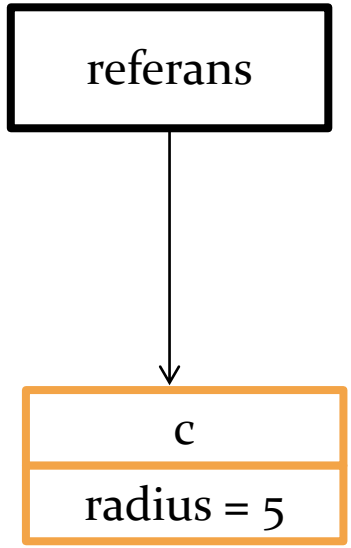
```
public class TestPassingObject {

    public static void main(String[] args) {
        Circle myCircle = new Circle(5.0);
        printCircle(myCircle);
    }

    static void printCircle(Circle c)
    {
        System.out.println("The area of the circle of radius " + c.radius+ "is "+c.findArea());
    }
}
```

Primitif tiplerle Nesneler arasındaki farklar:

- int i=1;
- i

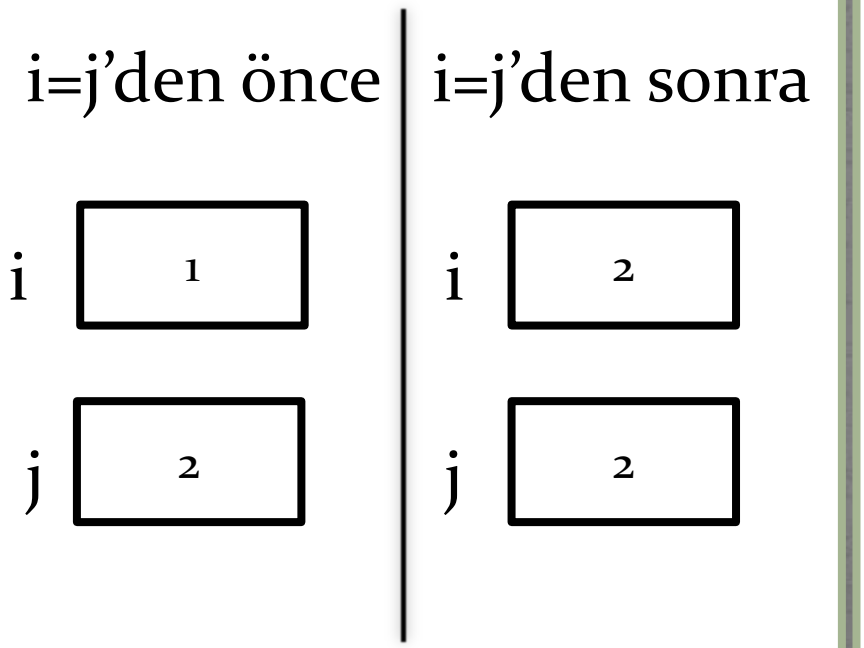


- Circle c = new Circle(5);
- c

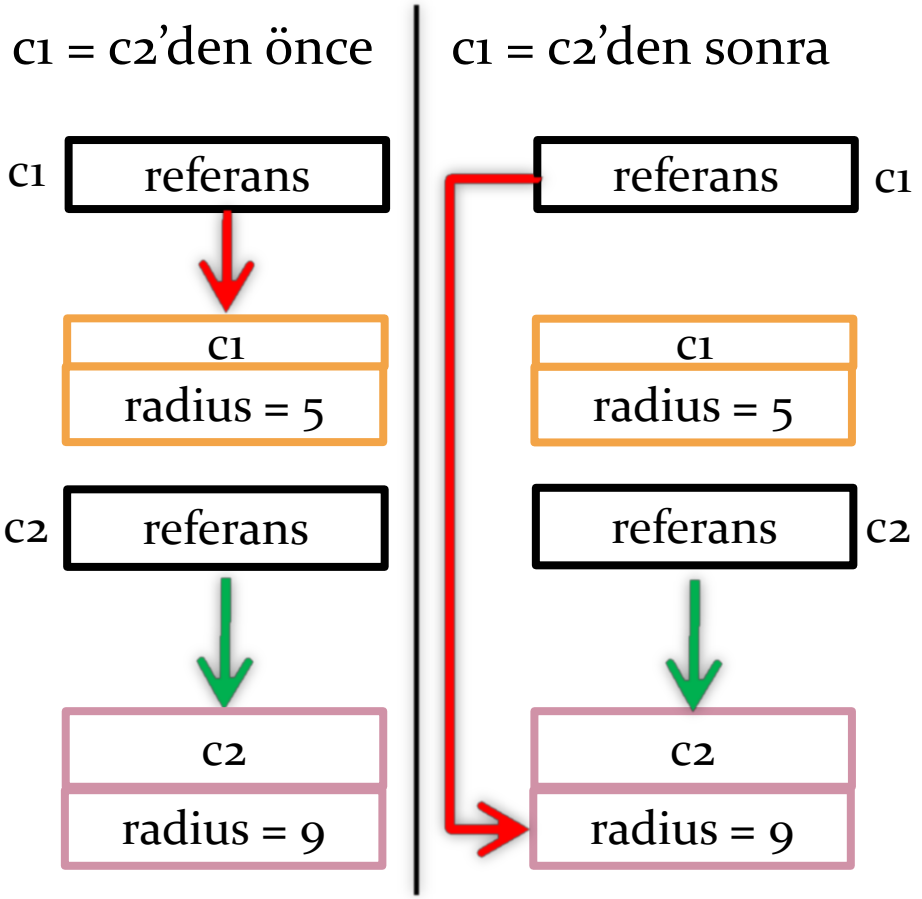


Primitif tiplerle Nesneler arasındaki farklar:

- int i=1;
- int j=2;



- Circle c1 = new Circle(5);
- Circle c2 = new Circle(9);

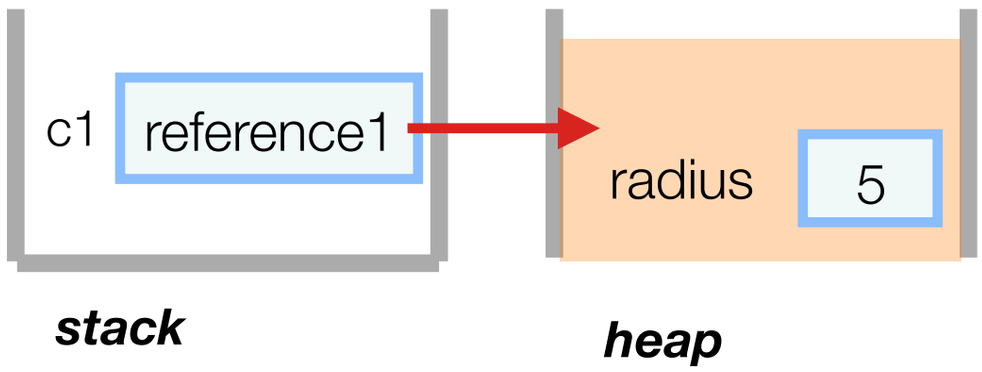


c1

Nesnelerin hafızada saklanması

Bir sınıfın nesnesi oluşturulduğunda nesnenin içerdiği veriler için heap'te alan açılır. stack'de ise heapteki bu alanı gösteren bir referans tutulur.

Circle c = new Circle(5)

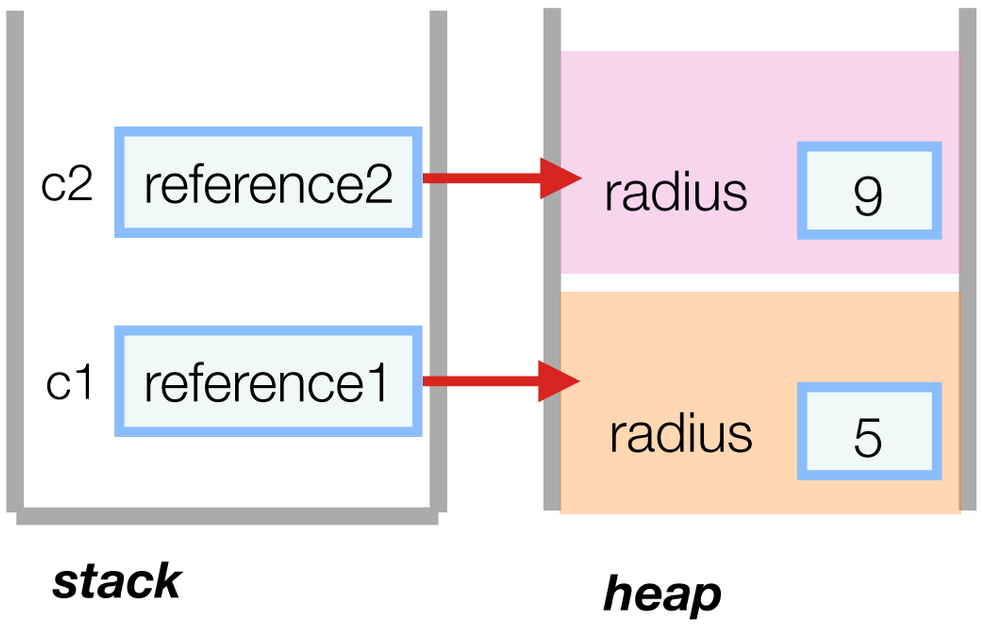


Nesnelerin hafızada saklanması

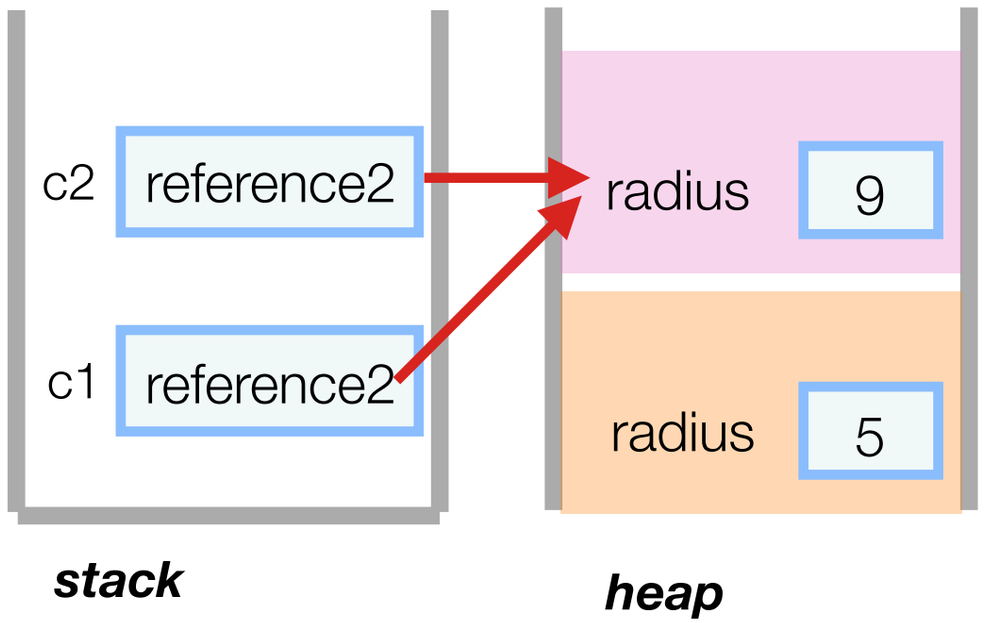
= operatörü nesnenin içeriğini değil, nesneye referans değeri kopyalar.

Circle c1 = new Circle(5)

Circle c2 = new Circle(9)



c1 = c2



Primitif tiplerle Nesneler arasındaki farklar:

Primitif tipler → Pass by value

Nesneler → Pass by reference

```
static void changeValues(Circle c, int x)
{
    x=5;
    c.radius = 5;
}
```

```
Circle myCircle = new Circle(7);
int num = 7;
```

=

```
myCircle.radius = 7.0
num = 7
```

```
changeValues(myCircle, num);
```

```
myCircle.radius = 5.0
num = 7
```

→ myCircle.radius değişti ancak num değişmedi.

GÖRÜNÜRLÜK NİTELEYİCİLERİ (VISIBILITY MODIFIERS)

- Java, verilere, metotlara ve sınıflara ulaşımı kontrol eden çeşitli niteleyiciler barındırır. Bunlardan public, private ve varsayılan niteleyiciyi göreceğiz.
  - public: Sınıf, metot ve verileri tüm programlar onlara erişebilecek biçimde niteler.
  - private: Metot ve verileri sadece tanımlandıkları sınıfta erişilebilecek biçimde niteler.
  - varsayılan niteleyici: Sınıf, metot ya da veriler tanımlanırken başlarına herhangi bir görünürlük niteleyicisi yazılmazsa, aynı paket içindeki herhangi bir sınıf tarafından erişilebilirdirler.

get ve set yöntemleri.

- Çoğu zaman bir sınıf değişkeninin değiştirilmesi ve sınıf değişkenine erişimde, sınıfı tasarlayanın kontrolünün olması gerekir.
- Örneğin her zaman pozitif değerli olması gereken bir sınıf değişkeni, public ile tanımlanıp direkt değiştirilebilir veya erişilebilir olursa, bu değişkene negatif değerler de atanabilir ve bu sınıfın diğer elemanlarını kötü yönde etkileyebilir.
- Bu sebeple
  - sınıf değişkenleri private olarak tanımlanır.
  - private değişkene ulaşmak için public bir get yöntemi yazılır.
  - private değişkeni değiştirmek için public bir set yöntemi yazılır.

```java
public class Circle {

    private double radius;

    public Circle() { radius = 1.0; }

    public Circle(double r) { setRadius(r); }

    public double getRadius() { return radius; }

    public void setRadius(double newRadius){
        if(newRadius<=0){
            System.out.println("Warning: Radius must be positive" +
                    ", it's set to default value 1 ");
            radius = 1;
        }else{
            radius = newRadius;
        }
    }

    public double findArea() { return radius*radius*Math.PI; }
}
```

Soru 1 - Karmaşık Sayılar

Karmaşık sayılar $z = a + b.i$ formunda tanımlanan sayılara verilen addır. Bu tanımda z karmaşık sayı, a sayının Reel bileşeni, b ise imajiner bileşeni olarak tanımlanır. İmajiner bileşeni temsil eden i değeri $i^2 = -1$ olarak tanımlanmıştır. Toplama işleminde karmaşık sayının reel ve imajiner bileşenleri ayrı ayrı toplanır, örnek $z_1 = a+bi$, $z_2 = c+di$ ise $z_1 + z_2 = (a+c)+(b+d)i$ olmaktadır. Çarpma işlemi $z_1 \times z_2 = (bd - ac) + (ad + ac)i$ şeklinde olmaktadır.