

# MTK467 Nesneye Yönelik Programlama

---

Ders 13 - Arayüzler, Generics

Zümra Kavafoğlu  
*<https://zumrakavafoglu.github.io/>*

- Bazı durumlarda bir sınıfın birden fazla sınıftan kalıt olması gerekebilir. Ancak biliyoruz ki bir sınıfın sadece bir adet süper sınıfı olabilir. Ancak bir sınıf birden fazla arayüzden kalıt olabilir.
- Arayüz sadece sabit değişkenler ve soyut metotlar içeren sınıf benzeri bir yapıdır.


```
public interface Comparable {  
    public int compareTo(Object o);  
}
```

## ARRAYÜZ (INTERFACE)

```
public class ComparableCircle extends Circle implements IComparable{  
  
    public ComparableCircle(double radius) {  
        super(radius);  
    }  
  
    @Override  
    public int compareTo(Object o) {  
        if (o instanceof ComparableCircle){  
            if (getRadius() > ((ComparableCircle) o).getRadius())  
                return 1;  
            else if (getRadius() < ((ComparableCircle) o).getRadius())  
                return -1;  
            else return 0;  
        }  
        else  
            return 0;  
    }  
}
```

Circle nesnelerini yarıçaplarına göre karşılaştıran overridden compareTo metodu


```
public class ComparableCylinder extends Cylinder implements IComparable {  
    public ComparableCylinder(double radius, double height) {  
        super(radius,height);  
    }  
  
    @Override  
    public int compareTo(Object o) {  
        if(o instanceof ComparableCylinder){  
            if(findVolume() > ((ComparableCylinder)o).findVolume())  
                return 1;  
            else if (findVolume() < ((ComparableCylinder)o).findVolume())  
                return -1;  
            else return 0;  
        }  
        else  
            return 0;  
    }  
}
```



Cylinder nesnelerini hacimlerine göre karşılaştıran overridden compareTo metodu

```
public class Max {
```

```
    public static IComparable max(IComparable o1, IComparable o2)
    {
        if(o1.compareTo(o2)>0)
            return o1;
        else
            return o2;
    }
}
```



IComparable arayüzü sayesinde, nesneler için genel bir maksimum bulma metodu tanımlanabilir. IComparable'ı implement eden bir sınıfın iki nesnesinden hangisinin en büyük olduğu, sınıf içinde override edilen compareTo metoduna göre bulunur.

circle1 ve circle2 nesnelerinden büyük olanı Circle sınıfında tanımlı compareTo() metodunu kullanarak bulur. Dolayısıyla max metodu yarıçapı büyük olan Circle nesnesini döndürür.

```
public class TestInterface {
```

```
    public static void main(String[] args) {
```

```
        ComparableCircle circle1 = new ComparableCircle(5);
```

```
        ComparableCircle circle2 = new ComparableCircle(4);
```

```
        IComparable circle = Max.max(circle1, circle2);
```

```
        System.out.println("The max circle's radius is " + ((Circle)circle).getRadius());
```

```
        System.out.println(circle);
```

```
        ComparableCylinder cylinder1 = new ComparableCylinder(5, 2);
```

```
        ComparableCylinder cylinder2 = new ComparableCylinder(4, 5);
```

```
        IComparable cylinder = Max.max(cylinder1, cylinder2);
```

```
        System.out.println("\ncylinder1's volume is "+cylinder1.findVolume());
```

```
        System.out.println("cylinder2's volume is "+cylinder2.findVolume());
```

```
        System.out.println("The max cylinder's \tradius is "
```

```
            + ((Cylinder)cylinder).getRadius()
```

```
            + "\n\t\ttheheight is "+((Cylinder)cylinder).getHeight()
```

```
            + "\n\t\t\tvolume is "+((Cylinder)cylinder).findVolume());
```

```
        System.out.println(cylinder);
```

```
    }
```

```
}
```

cylinder1 ve cylinder2 nesnelerinden büyük olanı Cylinder sınıfında tanımlı compareTo() metodunu kullanarak bulur. Dolayısıyla max metodu hacmi büyük olan Cylinder nesnesini döndürür.

*Bir interface yalnızca sabit veriler ve soyut metotlar içerebilir.*

```
public interface IMovable {
```

```
    public final double stepSize = 1;
```

```
    public void moveLeft();
```

```
    public void moveRight();
```

```
    public void moveUp();
```

```
    public void moveDown();
```

```
}
```



sabit stepSize verisi

```
public interface IMovable {  
  
    public final double stepSize = 1;  
  
    public void moveLeft();  
    public void moveRight();  
    public void moveUp();  
    public void moveDown();  
  
}
```



soyut metot deklarasyonları



*Bir sınıfın yalnızca bir adet süper sınıfı olabilir ama birden fazla Interface'i implement edebilir.*

```
public class CircleWithCenter extends Circle implements IMovable, IComparable{

    double centerX, centerY;

    public CircleWithCenter() { this(1,0,0); }

    public CircleWithCenter(double radius, double centerX, double centerY){

        super(radius);
        this.centerX = centerX;
        this.centerY = centerY;
    }

    @Override
    public int compareTo(Object o) {
        if(getRadius()>((Circle)o).getRadius())
            return 1;
        else if (getRadius()<((Circle)o).getRadius())
            return -1;
        else
            return 0;
    }

    @Override
    public void moveLeft() { centerX -= stepSize; }

    @Override
    public void moveRight() { centerX += stepSize; }

    @Override
    public void moveUp() { centerY += stepSize; }

    @Override
    public void moveDown() { centerY -= stepSize; }
}
```

İki interface implement ediyor.

*CircleWithCenter sınıfı hem IMovable hem de IComparable arayüzlerinin tüm metotlarını yeniden yazmalıdır (overriding).*

```
public class CircleWithCenter extends Circle implements IMovable, IComparable{
```

```
    double centerX, centerY;
```

```
    public CircleWithCenter() { this(1,0,0); }
```

```
    public CircleWithCenter(double radius, double centerX, double centerY){  
        super(radius);  
        this.centerX = centerX;  
        this.centerY = centerY;  
    }
```

```
@Override  
public int compareTo(Object o) {  
    if(getRadius()>((Circle)o).getRadius())  
        return 1;  
    else if (getRadius()<((Circle)o).getRadius())  
        return -1;  
    else  
        return 0;  
}
```

→ IComparable arayüzünün  
compareTo metodunun  
yeniden yazımı.

```
@Override  
public void moveLeft() { centerX -= stepSize; }  
  
@Override  
public void moveRight() { centerX += stepSize; }  
  
@Override  
public void moveUp() { centerY += stepSize; }  
  
@Override  
public void moveDown() { centerY -= stepSize; }  
}
```

→ IMovable arayüzünün  
metotlarının yeniden yazımı.

```
public class CylinderWithCenter extends Cylinder implements Comparable{
    double centerX, centerY;

    public CylinderWithCenter() { this(1,1,0,0); }

    public CylinderWithCenter(double radius, double height, double centerX, double centerY){
        super(radius, height);
        this.centerX = centerX;
        this.centerY = centerY;
    }

    @Override
    public int compareTo(Object o) {
        if(findVolume()>((Cylinder)o).findVolume())
            return 1;
        else if (findVolume()<((Cylinder)o).findVolume())
            return -1;
        else
            return 0;
    }
}
```

---

```
public class Point implements IMovable{
```

```
    protected double x;  
    protected double y;
```

```
    public Point() { this(0,0); }
```

```
    public Point(double x, double y){  
        this.x = x;  
        this.y = y;  
    }
```

```
    @Override
```

```
    public void moveLeft() { x -= stepSize; }
```

```
    @Override
```

```
    public void moveRight() { x += stepSize; }
```

```
    @Override
```

```
    public void moveUp() { y += stepSize; }
```

```
    @Override
```

```
    public void moveDown() { y -= stepSize; }
```

```
}
```

---

```

public class TestInterface2 {

    public static void main(String[] args) {

        CircleWithCenter circle1 = new CircleWithCenter(5, 1, 1);
        CircleWithCenter circle2 = new CircleWithCenter(4, 2, 2);

        IComparable circle = Max.max(circle1, circle2);
        System.out.println("The max circle's radius is " + ((Circle)circle).getRadius());
        System.out.println(circle);

        CylinderWithCenter cylinder1 = new CylinderWithCenter(5, 2, 3, 3);
        CylinderWithCenter cylinder2 = new CylinderWithCenter(4, 5, 4, 4);

        IComparable cylinder = Max.max(cylinder1, cylinder2);

        System.out.println("\ncylinder1's volume is "+cylinder1.findVolume());
        System.out.println("cylinder2's volume is "+cylinder2.findVolume());
        System.out.println("The max cylinder's \tradius is "
            + ((Cylinder)cylinder).getRadius()
            + "\n\t\ttheight is "+((Cylinder)cylinder).getHeight()
            + "\n\t\t\tvolume is "+((Cylinder)cylinder).findVolume());
        System.out.println(cylinder);

        Point point1 = new Point(1,2);
        Point point2 = new Point();

        IMovable[] movables = {circle1, circle2, point1, point2};

        for (IMovable movable: movables) {
            movable.moveDown();
        }
    }
}

```

# Generics

---

- Generics Java diliyle genel modeller oluşturmaya sağlayan bir konsepttir. Bu konsept sayesinde genel yani generic metotlar ve generic sınıflar yazabiliriz.
- Generic metotlar sayesinde tek bir metot tanımıyla birden fazla overloaded metodu temsil edebiliriz. Örneğin generic bir sort metodu yazıp, daha sonra bu metodu farklı tipte dizilerle çağırabiliriz. Benzer biçimde Generic sınıflar sayesinde de tek bir sınıf tanımıyla birden fazla ilişkili sınıf tanımlamış oluruz. Generic sınıflarla aynı mantıkla Generic arayüzler de tanımlanabilir.
- Generics konseptinin bir diğer faydası da derleme zamanı tip koruma(compile-time type safety) sağlamasıdır yani geçersiz tiplerin derleme zamanında yakalanmasını sağlar.

# Generic Metotlar: Motivasyon

- Overloaded metotlar
  - Farklı veri tipleri üzerinde benzer işlemler yaparlar. Örneğin Integer dizisi, Double dizisi ve Character dizisi ile printArray overloaded metodu:

```
1 // Fig. 18.1: OverloadedMethods.java
2 // Using overloaded methods to print array of different types.
3
4 public class OverloadedMethods
5 {
6     // method printArray to print Integer array
7     public static void printArray( Integer[] inputArray )
8     {
9         // display array elements
10        for ( Integer element : inputArray )
11            System.out.printf( "%s ", element );
12
13        System.out.println();
14    } // end method printArray
15
16    // method printArray to print Double array
17    public static void printArray( Double[] inputArray )
18    {
19        // display array elements
20        for ( Double element : inputArray )
21            System.out.printf( "%s ", element );
22
23        System.out.println();
24    } // end method printArray
25
```

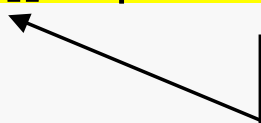
Method printArray accepts  
an array of Integer objects

Method printArray accepts  
an array of Double objects

# Generic Metotlar: Motivasyon

```
26 // method printArray to print Character array
27 public static void printArray( Character[] inputArray )
28 {
29     // display array elements
30     for ( Character element : inputArray )
31         System.out.printf( "%s ", element );
32
33     System.out.println();
34 } // end method printArray
35
36 public static void main( String args[] )
37 {
38     // create arrays of Integer, Double and Character
39     Integer[] integerArray = { 1, 2, 3, 4, 5, 6 };
40     Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
41     Character[] characterArray = { 'H', 'E', 'L', 'L', 'O' };
42
```

Method printArray accepts  
an array of Character objects





# Generic Metotlar: Motivasyon

```
43 System.out.println( "Array integerArray contains:" );
44 printArray( integerArray ); // pass an Integer array
45 System.out.println( "\nArray doubleArray contains:" );
46 printArray( doubleArray ); // pass a Double array
47 System.out.println( "\nArray characterArray contains:" );
48 printArray( characterArray ); // pass a Character array
49 } // end main
50 } // end class OverloadedMethods
```

```
Array integerArray contains:
1 2 3 4 5 6
```

```
Array doubleArray contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7
```

```
Array characterArray contains:
H E L L O
```

At compile time, the compiler determines argument `integerArray`'s type (i.e., `Integer[]`), attempts to locate a method named `printArray` that specifies a single `Integer[]` parameter (lines 7-14)

At compile time, the compiler determines argument `doubleArray`'s type (i.e., `Double[]`), attempts to locate a method named `printArray` that specifies a single `Double[]` parameter (lines 17-24)

At compile time, the compiler determines argument `characterArray`'s type (i.e., `Character[]`), attempts to locate a method named `printArray` that specifies a single `Character[]` parameter (lines 27-34)

# Generic Metotlar: Tip parametreleri

---

- Bu printArray metotları tek bir metotla temsil edilebilir:
  - Dizi tiplerini generic(genel) bir isimle değiştir.
  - Tek bir printArray metodu yaz
- Bu genel isme tip parametresi (type parameter) denir.
- Tip parametreleri,
  - dönüş tipi, parametre tipi ve lokal değişken tiplerini deklare etmek için kullanılabilir.
  - Generic metoda verilen argümanların tipleri için bir yer-tutucu görevi görür.
  - yalnızca referans veri tiplerini temsil edebilirler. Dolayısıyla primitif veri tiplerini temsil edemezler. (Bu sebeple int yerine Integer, double yerine Double vs. kullanmalıyız.)

```
public static < E > void printTwoArrays( E[ ] array1, E[ ] array2 )
```

# Generic Metotlar:

```
1 // Fig. 18.3: GenericMethodTest.java
2 // Using generic methods to print array of different types.
3
4 public class GenericMethodTest
5 {
6     // generic method printArray
7     public static < E > void printArray( E[] inputArray )
8     {
9         // display array elements
10        for ( E element : inputArray )
11            System.out.printf( "%s ", element );
12        System.out.println();
13    } // end method printArray
14
15
16    public static void main( String args[] )
17    {
18        // create arrays of Integer, Double and Character
19        Integer[] intArray = { 1, 2, 3, 4, 5 };
20        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
21        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
22    }
```

Use the type parameter to declare method `printArray`'s parameter type

Type parameter section delimited by angle brackets (< and > )

Use the type parameter to declare method `printArray`'s local variable type

# Generic Metotlar: Tip parametreleri

```
23      System.out.println( "Array integerArray contains:" );
24      printArray( integerArray ); // pass an Integer array
25      System.out.println( "\nArray doubleArray contains:" );
26      printArray( doubleArray ); // pass a Double array
27      System.out.println( "\nArray characterArray contains:" );
28      printArray( characterArray ); // pass a Character array
29  } // end main
30 } // end class GenericMethodTest
```

Invoke generic method `printArray` with an `Integer` array

Invoke generic method `printArray` with a `Double` array

Invoke generic method `printArray` with a `Character` array

```
Array integerArray contains:
1 2 3 4 5 6
```

```
Array doubleArray contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7
```

```
Array characterArray contains:
H E L L O
```

# Generic Metotlar: Sık karşılaşılan hatalar

---

- Generic metot tanımlarken dönüş tipinden önce tip parameter kısmını unutmak derleyici hatasına sebep olur.
- Derleyici bir metot çağırılışı için birden fazla uygun generic metotla karşılaşırsa hata verir.

# Generic Metotlar: Derleme zamanı çevirisi

- Derleme zamanı çevirisi (Compile-time translation)
- Derleyici printArray generic metodunu Java byte koduna çevirirken, tip parametresi kısmını siler ve tip parametrelerini gerçek tiplerle değiştirir. Bu işleme erasure(silme) denir. Default olarak bütün generic tipler Object tipiyle değiştirilir. printArray metodunun derlenmiş hali aşağıdaki gibi görünür. Bu kodun yalnızca bir kopyası vardır ve örneğimizdeki tüm printArray çağırılışlarında bu kopya kullanılır.

```
1 public static void printArray( Object[] inputArray )
2 {
3     // display array elements
4     for ( Object element : inputArray )
5         System.out.printf( "%s\n", element );
6
7     System.out.println();
8 } // end method printArray
```

Remove type parameter section and replace type parameter with actual type Object

Replace type parameter with actual type Object

# Overloading Generic Methods

---

- Generic metotlar aşağıdaki biçimlerde overload edilebilirler.
  - Başka generic metotlar tarafından
    - Aynı metot ismi ama farklı metot parametleri
  - Generic olmayan metotlar tarafından
    - Aynı metot ismi ve aynı parametre sayısı
- Derleyici bir metot çağırışıyla karşılaştığında
  - İlk önce tam olarak aynı metot ismi ve argüman tiplerine sahip metodu arar.
- Eğer bulamazsa birebir aynı olmayan ama yine de eşleştirilebilir metodu arar.

# Overloading Generic Methods

---

- Generic metotlar aşağıdaki biçimlerde overload edilebilirler.
  - Başka generic metotlar tarafından
    - Aynı metot ismi ama farklı metot parametleri
  - Generic olmayan metotlar tarafından
    - Aynı metot ismi ve aynı parametre sayısı
- Derleyici bir metot çağırışıyla karşılaştığında
  - İlk önce tam olarak aynı metot ismi ve argüman tiplerine sahip metodu arar.
- Eğer bulamazsa birebir aynı olmayan ama yine de eşleştirilebilir metodu arar.



# Overloading Generic Methods

```
// generic method printArray
public static < E > void printArray( E[] inputArray ){

    // display array elements
    for ( E element : inputArray )
        System.out.printf( "%s ", element );

    System.out.println();

} // end method printArray
```

```
// generic method printArray
public static < E > void printArray( E[] inputArray , int lowSubscript, int highSubscript){

    if(lowSubscript < 0 || highSubscript >= inputArray.length){
        System.out.println("Invalid arguments");
        return;
    }

    // display array elements
    for (int i= lowSubscript ; i <= highSubscript; i++)
        System.out.printf( "%s ", inputArray[i]);

    System.out.println();

} // end method printArray
```

```
public static void printArray(Character[] inputArray){

    for(int i=0; i<inputArray.length; i++){
        System.out.printf( "%s ", inputArray[i]);
        if( i % 2 == 1 )
            System.out.println();
    }

}
```

overloaded generic printArray methods

overloaded non-generic printArray method

# Overloading Generic Methods

---

```
public static void main( String args[] )
{
    // create arrays of Integer, Double and Character

    Integer[] integerArray = { 1, 2, 3, 4, 5 };

    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };

    Character[] characterArray = { 'H', 'E', 'L', 'L', 'O' };

    System.out.println( "Array integerArray contains:" );

    printArray( integerArray ); // pass an Integer array

    System.out.println( "\nArray doubleArray contains:" );
    printArray( doubleArray, 2, 4); // pass a Double array

    System.out.println( "\nArray characterArray contains:" );

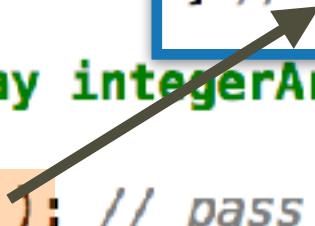
    printArray( characterArray ); // pass a Character array

} // end main
```

# Overloading Generic Methods

```
public static void main( String args[] ) {  
    // create arrays of Integer, Double, and Character  
    Integer[] integerArray = { 1, 2, 3, 4, 5 };  
    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5 };  
    Character[] characterArray = { 'A', 'B', 'C', 'D', 'E' };  
    System.out.println( "Array integerArray contains:" );  
    printArray( integerArray ); // pass an Integer array  
  
    System.out.println( "\nArray doubleArray contains:" );  
    printArray( doubleArray, 2, 4 ); // pass a Double array  
  
    System.out.println( "\nArray characterArray contains:" );  
    printArray( characterArray ); // pass a Character array  
}  
// end main
```

```
// generic method printArray  
public static < E > void printArray( E[] inputArray ) {  
    // display array elements  
    for ( E element : inputArray )  
        System.out.printf( "%s ", element );  
    System.out.println();  
} // end method printArray
```



# Overloading Generic Methods

```
public static void main( String args[] )
{
    // create an Integer array
    Integer[] intArray = {1, 2, 3, 4, 5};

    // create a Double array
    Double[] doubleArray = {1.1, 2.2, 3.3, 4.4, 5.5};

    // create a Character array
    Character[] characterArray = {'A', 'B', 'C', 'D', 'E'};

    System.out.println("Integer array contains:");
    printArray( intArray ); // pass an Integer array

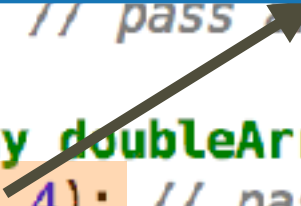
    System.out.println( "\nArray doubleArray contains:" );
    printArray( doubleArray, 2, 4); // pass a Double array

    System.out.println( "\nArray characterArray contains:" );
    printArray( characterArray ); // pass a Character array
} // end main
```

```
// generic method printArray
public static < E > void printArray( E[] inputArray , int lowSubscript, int highSubscript){
    if(lowSubscript < 0 || highSubscript >= inputArray.length){
        System.out.println("Invalid arguments");
        return;
    }

    // display array elements
    for (int i= lowSubscript ; i <= highSubscript; i++)
        System.out.printf( "%s ", inputArray[i]);

    System.out.println();
} // end method printArray
```



# Overloading Generic Methods

```
public static void main( String args[] )
{
    // create arrays of Integer, Double and Character

    Integer[] integerArray = { 1, 2, 3, 4, 5 };

    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8 };

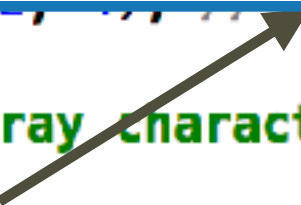
    Character[] characterArray = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J' };

    printArray( integerArray );

    System.out.println( "Double array contains:" );
    printArray( doubleArray );

    System.out.println( "\nArray characterArray contains:" );
    printArray( characterArray ); // pass a Character array
} // end main
```

```
public static void printArray(Character[] inputArray){
    for(int i=0; i<inputArray.length; i++){
        System.out.printf( "%s ", inputArray[i]);
        if( i % 2 == 1)
            System.out.println();
    }
}
```



# Overloading Generic Methods

---

```
Array integerArray contains:
```

```
1 2 3 4 5
```

```
Array doubleArray contains:
```

```
3.3 4.4 5.5
```

```
Array characterArray contains:
```

```
H E
```

```
L L
```

```
0
```



# Generic Sınıflar

- Bir sınıfı tipten bağımsız tanımlamayı sağlar. Daha sonra bu sınıf kullanılarak tipe özel nesneler tanımlanabilir.

```
public class GenericBox<E> {  
    // Private variable  
    private E content;  
  
    // Constructor  
    public GenericBox(E content) {  
        this.content = content;  
    }  
  
    public E getContent() {  
        return content;  
    }  
  
    public void setContent(E content) {  
        this.content = content;  
    }  
  
    public String toString() {  
        return content + " (" + content.getClass() + ")";  
    }  
}
```

Generic class declaration, class name is followed by a type parameter section

# Generic Sınıflar: GenericBox örneği

---

```
public class TestGenericBox {  
    public static void main(String[] args) {  
        GenericBox<String> box1 = new GenericBox<String>("Hello");  
        String str = box1.getContent(); // no explicit downcasting needed  
        System.out.println(box1);  
        GenericBox<Integer> box2 = new GenericBox<Integer>(123);  
        int i = box2.getContent();      // downcast to Integer, auto-unbox to int  
        System.out.println(box2);  
        GenericBox<Double> box3 = new GenericBox<Double>(55.66);  
        double d = box3.getContent();   // downcast to Double, auto-unbox to double  
        System.out.println(box3);  
    }  
}
```



# Generic Sınıflar: Stack örneği

```
public class Stack<E> {  
  
    private final int size; // number of elements in the stack  
  
    private int top; // location of the top element  
  
    private E[] elements; // array that stores stack elements  
  
    // no-argument constructor creates a stack of the default size  
  
    public Stack() {  
        this(10); // default stack size  
    } // end no-argument Stack constructor  
  
    public Stack( int s )  
    {  
        size = s > 0 ? s : 10; // set size of Stack  
  
        top = -1; // Stack initially empty  
        elements = ( E[] ) new Object[ size ]; // create array  
    } // end Stack constructor  
  
    // push element onto stack; if successful, return true;
```

# Generic Sınıflar: Stack örneği

---

```
public void push( E pushValue )
{
    if ( top == size - 1 ){ // if stack is full
        System.out.println("Stack is full, cannot push %s" + pushValue );
    }
    else
        elements[ ++top ] = pushValue; // place pushValue on Stack
} // end method push

// return the top element if not empty; else throw EmptyStackException

public E pop()
{
    if ( top == -1 ) { // if stack is empty
        System.out.println("Stack is empty, cannot pop");
        return null;
    }

    return elements[ top-- ]; // remove and return top element of Stack
} // end method pop
} // end class Stack< E >
```

# Generic Sınıflar: StackTest

```
public class StackTest
{
    public static void main(String[] args){

        double[] doubleElements = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
        int[] integerElements = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};

        Stack<Double> doubleStack; // stack stores Double objects
        Stack<Integer> integerStack; // stack stores Integer objects

        doubleStack = new Stack<Double>(5); // Stack of Doubles

        integerStack = new Stack<Integer>(10); // Stack of Integers

        pushDoubleArray(doubleStack, doubleElements);
        pushIntegerArray(integerStack, integerElements);

    }

    public static void pushDoubleArray( Stack<Double> stack, double[] elements){

        for(double element : elements){
            System.out.println("Element to push: " + element);
            stack.push(element);
        }

    }

    public static void pushIntegerArray( Stack<Integer> stack, int[] elements){

        for(int element : elements){
            System.out.println("Element to push: " + element);
            stack.push(element);
        }

    }

}
```

# Generic Sınıflar: StackTest

---

```
Element to push: 1.1
Element to push: 2.2
Element to push: 3.3
Element to push: 4.4
Element to push: 5.5
Element to push: 6.6
Stack is full, cannot push 6.6
Element to push: 1
Element to push: 2
Element to push: 3
Element to push: 4
Element to push: 5
Element to push: 6
Element to push: 7
Element to push: 8
Element to push: 9
Element to push: 10
Element to push: 11
Stack is full, cannot push 11
```

# Generic Siniflar: StackTestWithGenericMethod

---

```
public class StackTestWithGenericMethod {  
    public static void main(String[] args){  
        Double[] doubleElements = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};  
        Integer[] integerElements = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};  
  
        Stack<Double> doubleStack; // stack stores Double objects  
        Stack<Integer> integerStack; // stack stores Integer objects  
  
        doubleStack = new Stack<Double>(5); // Stack of Doubles  
        integerStack = new Stack<Integer>(10); // Stack of Integers  
  
        pushArray(doubleStack, doubleElements);  
        pushArray(integerStack, integerElements);  
    }  
  
    public static <T> void pushArray( Stack<T> stack, T[] elements){  
        for(T element : elements){  
            System.out.println("Element to push: " + element);  
            stack.push(element);  
        }  
    }  
}
```

# Birden fazla tip parametrelili Generic Sınıflar: Pair ve Map

---

- Belirli tipte bir değeri (key) belirli tipte başka bir değeri (value) eşleyen yapılara Map denir.
- Mapte her bir keyden yalnızca bir tane bulunabilir.
- Kendi generic MyMap sınıfımızı yazalım. key ve value farklı tiplerde olabileceğinden MyMap sınıfı iki tip parametrelili bir sınıf olmalıdır.
- MyMap sınıfında kullanmak üzere iki tip parametrelili Pair sınıfını yazalım. Pair sınıfı bir (key,value) ikilisini temsil etsin.

# Birden fazla tip parametrelili Generic Sınıflar: Pair

```
public class Pair<T ,S> {  
  
    private T key;  
    private S value;  
  
    private Pair<T,S> next;  
  
    public Pair(T key, S value) {  
        this.key = key;  
        this.value = value;  
    }  
  
    public T getKey() { return key; }  
  
    public void setKey(T key) { this.key = key; }  
  
    public S getValue() { return value; }  
  
    public void setValue(S value) { this.value = value; }  
  
    public void setNext(Pair<T, S> next) { this.next = next; }  
  
    public Pair<T,S> getNext() { return next; }  
  
}
```

Mapi bağlantılı bir liste olarak oluşturmak için her pair kendinden sonraki pair bilgisini tutar.



# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

```
public class MyMap<T, S> {  
    private Pair<T, S> firstPair;  
}
```



**Map'i pairların bağlantılı bir listesi şeklinde oluşturacağız.**  
**Her pair kendinden bir sonraki pair'ı next verisinde tuttuğu için,**  
**mapin yalnızca ilk pair'ını tutmamız yeterli.**



# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

key anahtarına sahip bir pair varsa döndürür yoksa null döndürür.

```
} private Pair<T, S> getPair(T key){  
    if(key == null){  
        System.out.println("Parameter key is null");  
        return null;  
    }  
  
    Pair<T, S> pair = firstPair;  
    while (pair != null){  
        if(pair.getKey().equals(key)){  
            return pair;  
        }  
  
        pair = pair.getNext();  
    }  
  
    return null;  
}
```

# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

mapte key anahtarına sahip bir değer varsa value ile günceller, yoksa (key,value) pair'ını map'e ekler.

```
public void put(T key, S value){  
    if(this.isEmpty()){  
        firstPair = new Pair<T, S>(key, value);  
    }  
    else{  
        Pair<T, S> pair = this.getPair(key);  
  
        if(pair != null){  
            pair.setValue(value);  
        }else{  
            this.getLast().setNext(new Pair<T, S>(key, value));  
        }  
    }  
}
```

# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

mapteki son pair'ı döndürür

```
private Pair<T,S> getLast(){  
    if(this.isEmpty())  
        return null;  
  
    Pair<T,S> lastPair = firstPair;  
  
    while(lastPair.getNext() != null){  
        lastPair = lastPair.getNext();  
    }  
  
    return lastPair;  
}
```

# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

mapte key anahtarına karşılık gelen bir değer varsa döndürür.

```
public S get(T key){  
    Pair<T, S> pair = this.getPair(key);  
  
    if(pair != null)  
        return pair.getValue();  
  
    return null;  
}
```

# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

map boşsa true döner

```
public boolean isEmpty() { return firstPair == null; }
```

# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

Parametre olarak verilen pair'dan bir önceki pairı döndürür.

```
private Pair<T, S> getPrevious(Pair<T, S> pair){  
    if(pair == null)  
        return null;  
  
    Pair<T, S> tempPair = firstPair;  
  
    while (tempPair != null){  
        if(tempPair.getNext() == pair)  
            return tempPair;  
  
        tempPair = tempPair.getNext();  
    }  
  
    return null;  
}
```

# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

Anahtarı key olan bir pair varsa onu siler ve değerini döndürür.

```
public S remove(T key){  
    Pair<T, S> pair = this.getPair(key);  
  
    if(pair != null){  
        this.getPrevious(pair).setNext(pair.getNext());  
        return pair.getValue();  
    }  
  
    System.out.println("Cannot remove");  
    return null;  
}
```

# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

**Mapi boş hale getirir.**

```
public void clear() { firstPair = null; }
```



# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

map key anahtarlı bir pair'a sahipse true döner.

```
public boolean containsKey(T key) {  
    return this.getPair(key) != null;  
}
```

# Birden fazla tip parametrelili Generic Sınıflar: MyMap

---

```
public String toString(){  
    System.out.println("Map: ");  
    if(isEmpty())  
        return "Empty Map";  
    String output = "";  
    Pair<T, S> pair = firstPair;  
    while (pair != null){  
        output += pair;  
        pair = pair.getNext();  
    }  
    return output;  
}
```

# MyMaptest: PersonMap

```
import java.util.Scanner;

public class MyMapTest {

    public static void main(String[] args){

        MyMap<Long, Person> personMap = new MyMap<Long, Person>();

        personMap.put(64783912567L, new Person("Ayse", "Demir"));
        personMap.put(19832166541L, new Person("Hale", "Berber"));
        personMap.put(988786654332L, new Person("Kemal", "Ark"));
        personMap.put(901231234516L, new Person("Fatma", "Kale"));

        System.out.println("Enter the citizen id number: ");
        Scanner input = new Scanner(System.in);

        Long idNumber = input.nextLong();

        if(personMap.containsKey(idNumber)){
            System.out.println("Citizen with id " + idNumber + " is " + personMap.get(idNumber));
        }else{
            System.out.println("No citizen with id " + idNumber);
        }
    }
}
```

Long tipinde anahtarları Person tipinde değerlerle eşleştiren personMap nesnesi

# MyMaptest: PersonMap

---

## Çıktı

```
Enter the citizen id number:
```

```
988786654332
```

```
Citizen with id 988786654332 is Kemal Ark
```