

BBS515 Nesneye Yönelik Programlama

Ders 12 - Generics

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

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
```

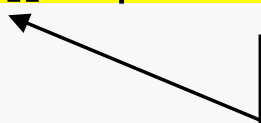
Method printArray accepts
an array of Integer objects

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\n", element );
12
13        System.out.println();
14    } // end method printArray
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
```

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

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

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

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

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

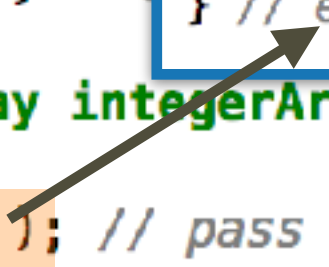
```
public static void main( String args[] ) {
    // create arrays of Integer, Double, and Character
    Integer[] integerArray = { 1, 2, 3, 4, 5 };
    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    Character[] characterArray = { 'A', 'B', 'C', 'D', 'E' };

    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
```



A diagram consisting of a black arrow pointing from the `printArray(integerArray);` call in the `main` method to the `printArray` method definition in the inset box.

Overloading Generic Methods

```
public static void main( String args[] )
{
    // create an Integer array
    Integer[] intArray = new Integer[10];
    // create a Double array
    Double[] doubleArray = new Double[10];
    // create a Character array
    Character[] characterArray = new Character[10];

    System.out.println( "\nArray integerArray 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, 1.1 };

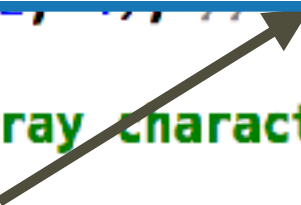
    Character[] characterArray = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J' };

    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 Sınıf: 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 değere (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
```


MyMaptest2: Address sınıfı

```
public class Address {  
  
    private int houseNumber;  
    private String street;  
    private String city;  
    private String country;  
  
    Address(int houseNumber, String street, String city, String country){  
        this.houseNumber = houseNumber;  
        this.street = street;  
        this.city = city;  
        this.country = country;  
    }  
  
    public int getHouseNumber() {  
        return houseNumber;  
    }  
  
    public void setHouseNumber(int houseNumber) {  
        this.houseNumber = houseNumber;  
    }  
  
    public String getStreet() {  
        return street;  
    }  
  
    public void setStreet(String street) {  
        this.street = street;  
    }  
}
```

MyMaptest2: Address sınıfı (devamı)

```
public String getCity() {  
    return city;  
}  
  
public void setCity(String city) {  
    this.city = city;  
}  
  
public String getCountry() {  
    return country;  
}  
  
public void setCountry(String country) {  
    this.country = country;  
}  
  
@Override  
public String toString() {  
    return street + " No: " + houseNumber + " - " + city + " / " + country ;  
}
```

MyMaptest2: AddressPhoneMap

```
public class MyMapTest2 {  
    public static void main(String[] args){  
        MyMap<Address, Long> addressPhoneMap = new MyMap<Address, Long>();  
  
        addressPhoneMap.put(new Address(13, "Sumak Sk.", "Ankara", "Turkiye"), 3122112321L);  
        addressPhoneMap.put(new Address(2, "Moda Sk.", "Istanbul", "Turkiye"), 2123458989L);  
        addressPhoneMap.put(new Address(13, "Sumak Sk.", "Ankara", "Turkiye"), 3126572321L);  
  
        System.out.println(addressPhoneMap);  
    }  
}
```

MyMaptest2: AddressPhoneMap

```
public class MyMapTest2 {  
  
    public static void main(String[] args){  
  
        MyMap<Address, Long> addressPhoneMap = new MyMap<Address, Long>();  
  
        addressPhoneMap.put(new Address(13, "Sumak Sk.", "Ankara", "Türkiye"), 3122112321L);  
        addressPhoneMap.put(new Address(2, "Moda Sk.", "İstanbul", "Türkiye"), 2123458989L);  
        addressPhoneMap.put(new Address(13, "Sumak Sk.", "Ankara", "Türkiye"), 3126572321L);  
  
        System.out.println(addressPhoneMap);  
  
    }  
}
```



Aynı anahtar değerli iki girdi map'e eklenirse, bu anahtara karşılık gelen değer güncellenir.

MyMaptest2: AddressPhoneMap

```
public class MyMapTest2 {  
  
    public static void main(String[] args){  
  
        MyMap<Address, Long> addressPhoneMap = new MyMap<Address, Long>();  
  
        addressPhoneMap.put(new Address(13, "Sumak Sk.", "Ankara", "Türkiye"), 3122112321L);  
        addressPhoneMap.put(new Address(2, "Moda Sk.", "İstanbul", "Türkiye"), 2123458989L);  
        addressPhoneMap.put(new Address(13, "Sumak Sk.", "Ankara", "Türkiye"), 3126572321L);  
  
        System.out.println(addressPhoneMap);  
  
    }  
}
```

Ama çıktıda görüldüğü gibi map iki anahtarı farklı kabul etmiş, dolayısıyla mapde aynı anahtara sahip iki girdi var.

Aynı anahtar değerli iki girdi map'e eklenirse, bu anahtara karşılık gelen değer güncellenir.

Çıktı

Map:

[Sumak Sk. No: 13 – Ankara / Türkiye, 3122112321]
[Moda Sk. No: 2 – İstanbul / Türkiye, 2123458989]
[Sumak Sk. No: 13 – Ankara / Türkiye, 3126572321]

MyMaptest2: AddressPhoneMap

```
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));  
        }  
    }  
}
```

MyMaptest2: AddressPhoneMap

```
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;  
}
```


Overriding equals method

Bir sınıfın equals yöntemi override edilmezse == ile aynı biçimde çalışır yani nesnenin içeriğini değil hafızadaki yerini karşılaştırır.

```
public class TestEquality1 {  
    public static void main(String[] args){  
        MyMap<Address, Long> addressPhoneMap = new MyMap<Address, Long>();  
  
        Address address1 = new Address(13, "Sumak Sk.", "Ankara", "Türkiye");  
        Address address2 = new Address(13, "Sumak Sk.", "Ankara", "Türkiye");  
  
        System.out.println("Adress1: "+address1);  
        System.out.println("Adress2: "+address2);  
  
        if(address1.equals(address2))  
            System.out.println("Adress1 is same with address2");  
        else  
            System.out.println("Adress2 is not same with address2");  
    }  
}
```

```
Adress1: Sumak Sk. No: 13 – Ankara / Türkiye  
Adress2: Sumak Sk. No: 13 – Ankara / Türkiye  
Adress2 is not same with address2
```


Overriding equals method

- equals metodunun içeriği aynı iki Adress nesnesi için true döndürmesi için, Address sınıfının equals metodunu override etmeliyiz.
- Ancak bir kural olarak, bir sınıfın equals metodunu override ettiğimiz her zaman o sınıfın hashCode() metodunu da override etmeliyiz.
- hashcode javanın her bir nesne için ürettiği bir tamsayıdır. hashCode() metodunu override etmek için Objects sınıfının hash metodu kullanılabilir.

Overriding equals method

```
@Override
public boolean equals(Object o) {
    if (this == o)
        return true;
    if (o == null || getClass() != o.getClass())
        return false;

    Address address = (Address) o;

    if (houseNumber != address.houseNumber)
        return false;
    if (!street.equals(address.street))
        return false;
    if (!city.equals(address.city))
        return false;
    if (!country.equals(address.country))
        return false;

    return true;
}

@Override
public int hashCode() {
    return Objects.hash(houseNumber, street, city, country);
}
```

Overriding equals method

- Address sınıfının equals ve hashCode metotlarının override edilmesiyle istediğimiz sonuca ulaşırız.

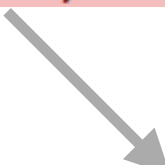
```
public class TestEquality1 {  
    public static void main(String[] args){  
        MyMap<Address, Long> addressPhoneMap = new MyMap<Address, Long>();  
  
        Address address1 = new Address(13, "Sumak Sk.", "Ankara", "Turkiye");  
        Address address2 = new Address(13, "Sumak Sk.", "Ankara", "Turkiye");  
  
        System.out.println("Adress1: "+address1);  
        System.out.println("Adress2: "+address2);  
  
        if(address1.equals(address2))  
            System.out.println("Adress1 is same with address2");  
        else  
            System.out.println("Adress2 is not same with address2");  
    }  
}
```

```
Adress1: Sumak Sk. No: 13 - Ankara / Turkiye  
Adress2: Sumak Sk. No: 13 - Ankara / Turkiye  
Adress1 is same with address2
```

Overriding equals method

- Address sınıfının equals ve hashCode metotlarının override edilmesiyle istediğimiz sonuca ulaşırız.

```
public class MyMapTest2 {  
    public static void main(String[] args){  
        MyMap<Address, Long> addressPhoneMap = new MyMap<Address, Long>();  
  
        addressPhoneMap.put(new Address(13, "Sumak Sk.", "Ankara", "Türkiye"), 3122112321L);  
        addressPhoneMap.put(new Address(2, "Moda Sk.", "İstanbul", "Türkiye"), 2123458989L);  
        addressPhoneMap.put(new Address(13, "Sumak Sk.", "Ankara", "Türkiye"), 3126572321L);  
  
        System.out.println(addressPhoneMap);  
    }  
}
```



Aynı anahtar değerli iki girdi map'e eklenirse, bu anahtara karşılık gelen değer güncellenir.

Map:

```
[ Sumak Sk. No: 13 – Ankara / Türkiye, 3126572321 ]  
[ Moda Sk. No: 2 – İstanbul / Türkiye, 2123458989 ]
```

Generic Sınıflar: Ham tipler (Raw types)

- Herhangi bir tip argümanı belirlemeden bir generic sınıf nesnesi oluşturmayı sağlar.

```
Stack objectStack = new Stack( 5 );
```

- Tip argümanı belirli bir Generic sınıf nesnesi, ham bir generic sınıf nesnesine atanabilir.

```
Stack rawTypeStack2 = new Stack<Double>(5);
```

- Tam tersi de yapılabilir ama güvenli değildir. Örneğin aşağıdaki örnekte sağ tarafta oluşturulan Stack tipi Integer olmayan elemanlar da içerebilir.

```
Stack<Integer> integerStack = new Stack(10);
```

Generic Sınıflar: Ham tipler (Raw types)

```
public class RawTypeTest {  
    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 rawTypeStack1 = new Stack(5);  
  
        Stack rawTypeStack2 = new Stack<Double>(5);  
  
        Stack<Integer> integerStack = new Stack(10); // stack stores Integer objects  
  
        StackTestWithGenericMethod.pushArray(rawTypeStack1, doubleElements);  
        StackTestWithGenericMethod.pushArray(rawTypeStack2, doubleElements);  
        StackTestWithGenericMethod.pushArray(integerStack, integerElements);  
    }  
}
```

Generic Arayüz

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

Generic olmayan IComparable arayüzü

Circle IComparable arayüzünü implement eder

```
public class Circle extends GeometricObject implements IComparable
```

Circle sınıfının compareTo metodu

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

```
public class ComparableTest {  
    public static void main(String[] args){  
        Circle c1 = new Circle(3.0);  
        Rectangle r1 = new Rectangle(2,3);  
        System.out.print(c1.compareTo(r1));  
    }  
}
```

```
Exception in thread "main" java.lang.ClassCastException: Rectangle cannot be cast to Circle  
    at Circle.compareTo(Circle.java:49)  
    at ComparableTest.main(ComparableTest.java:11)
```

Rectangle nesnesi Circle sınıfına cast edilemeyeceğinden çalışma zamanında hata verir.

Generic Arayüz

```
public interface GenericComparable <T> {  
    public int compareTo(T other);  
}
```

T tip parametrelili Generic arayüz

Circle2, GenericComparable<Circle2> arayüzünü implement eder

```
public class Circle2 extends GeometricObject implements GenericComparable<Circle2>
```

```
@Override  
public int compareTo(Circle2 other) {  
    if (getRadius() > other.getRadius())  
        return 1;  
    else if (getRadius() < other.getRadius())  
        return -1;  
    else  
        return 0;  
}
```

Circle2 sınıfının compareTo metodu

```
public class ComparableTest2 {  
    public static void main(String[] args){  
        Circle2 c2 = new Circle2(3.0);  
        Rectangle r1 = new Rectangle(2,3);  
        System.out.print(c2.compareTo(r1));  
    }  
}
```

compareTo (Circle2) in Circle2 cannot be applied
to (Rectangle)

type-safety: Yanlış tipte argüman
derleme zamanı hatası verir

Tip parametreleri için üst sınır

- Tip parametrelerinin temsil edeceği değişken tiplerini kısıtlamak için bu parametrelere bir üst sınır konulabilir. Burada üst sınırdan kasıt bir süper sınıf veya arayüzdür. Eğer herhangi bir üst sınır belirtilmediyse üst sınır Object sınıfıdır.
- Örneğin tip parametresi T'nin yalnızca SClass süper sınıfının çocuklarını temsil etmesi isteniyorsa o zaman tip parametresi kısmına

< T extends SClass >

yazılır. Eğer SClass bir arayüzse ve T'nin yalnızca SClass arayüzünü implement eden sınıfları temsil etmesi isteniyorsa yine aynı gösterim kullanılır.

- Eğer T parametresi A sınıfının alt sınıfı olacak ve B ve C arayüzlerini implement edecek biçimde kısıtlanmak isteniyorsa

< T extends A & B & C >

yazılır. Burada A'nın ilk yazılmaması derleme-zamanı hatasına sebep olur.

Tip parametreleri için üst sınır

Comparable<T> Java'nın sağladığı bir arayüzdür. Bu arayüzü implement eden sınıfların **compareTo** metodunu implement etmesi gerekir.

maximum isimli Generic metodun yalnızca **Comparable<T>** arayüzünü implement eden tiplere sahip değişkenlerle çağırılabilmesini sağlar

```
public class MaximumTest {  
    // determines the largest of three Comparable objects  
    public static < T extends Comparable< T > > T maximum( T x, T y, T z )  
    {  
        T max = x; // assume x is initially the largest  
        if ( y.compareTo( max ) > 0 )  
            max = y; // y is the largest so far  
        if ( z.compareTo( max ) > 0 )  
            max = z; // z is the largest  
        return max; // returns the largest object  
    } // end method maximum  
  
    public static void main( String args[] ){  
        System.out.printf( "Maximum of %d, %d and %d is %d\n\n", 3, 4, 5, maximum(3,4,5));  
        System.out.printf( "Maximum of %.1f, %.1f and %.1f is %.1f\n\n", 6.6, 8.8, 7.7, maximum(6.6, 7.7, 8.8));  
        System.out.printf( "Maximum of %s, %s and %s is %s\n", "pear","apple", "orange", maximum("pear","apple", "orange"));  
    }  
}
```

Böylece her argüman nesnenin **compareTo** metodunun çağırılabilmesi garantilenmiş olur.

Arrays sınıfı

- Arrays sınıfı dizileri işlemek için static metotlar sağlayan bir Java.util sınıfıdır.
- Arrays sınıfı metotları:
 - sort : diziyi sıralar
 - binarySearch : sıralanmış bir dizide arama yapar
 - equals: dizileri karşılaştırır
 - fill: diziyi elemanlarla doldurur.

Arrays örneği

```
3  import java.util.Arrays;
4
5  public class UsingArrays
6  {
7      private int intArray[] = { 1, 2, 3, 4, 5, 6 };
8      private double doubleArray[] = { 8.4, 9.3, 0.2, 7.9, 3.4 };
9      private int filledIntArray[], intArrayCopy[];
10
11     // constructor initializes arrays
12     public UsingArrays()
13     {
14         filledIntArray = new int [ 10 ]; // create int array with 10 elements
15         intArrayCopy = new int [ intArray.length ];
16
17         Arrays.fill( filledIntArray, 7 ); // fill with 7s
18         Arrays.sort( doubleArray ); // sort doubleArray ascending
19
20         // copy array intArray into array intArrayCopy
21         System.arraycopy( intArray, 0, intArrayCopy,
22             0, intArray.length );
23     } // end UsingArrays constructor
```

Arrays örneği

```
47      // find value in array intArray
48      public int searchForInt( int value )
49      {
50          return Arrays.binarySearch( intArray, value );
51      } // end method searchForInt
52
53      // compare array contents
54      public void printEquality()
55      {
56          boolean b = Arrays.equals( intArray, intArrayCopy );
57          System.out.printf( "intArray %s intArrayCopy\n",
58                          ( b ? "==" : "!=" ) );
59
60          b = Arrays.equals( intArray, filledIntArray );
61          System.out.printf( "intArray %s filledIntArray\n",
62                          ( b ? "==" : "!=" ) );
63      } // end method printEquality
```

Arrays örneği

```
26     public void printArrays()  
27     {  
28         System.out.print( "doubleArray: " );  
29         for ( double doubleValue : doubleArray )  
30             System.out.printf( "%.1f ", doubleValue );  
31  
32         System.out.print( "\nintArray: " );  
33         for ( int intValue : intArray )  
34             System.out.printf( "%d ", intValue );  
35  
36         System.out.print( "\nfilledIntArray: " );  
37         for ( int intValue : filledIntArray )  
38             System.out.printf( "%d ", intValue );  
39  
40         System.out.print( "\nintArrayCopy: " );  
41         for ( int intValue : intArrayCopy )  
42             System.out.printf( "%d ", intValue );  
43  
44         System.out.println( "\n" );  
45     } // end method printArrays
```


Arrays örneği

```
65     public static void main( String args[] )
66     {
67         UsingArrays usingArrays = new UsingArrays();
68
69         usingArrays.printArrays();
70         usingArrays.printEquality();
71
72         int location = usingArrays.searchForInt( 5 );
73         if ( location >= 0 )
74             System.out.printf(
75                 "Found 5 at element %d in intArray\n", location );
76         else
77             System.out.println( "5 not found in intArray" );
78
79         location = usingArrays.searchForInt( 8763 );
80         if ( location >= 0 )
81             System.out.printf(
82                 "Found 8763 at element %d in intArray\n", location );
83         else
84             System.out.println( "8763 not found in intArray" );
85     } // end main
86 } // end class UsingArrays
```