CMPE 101
Object Oriented Programming

İstanbul
Bilgi University

Dr. Emel Küpçü

Department of Computer Engineering

İstanbul Bilgi University

Week-11: Generic Classes and Methods

Generics

- Generics in Java allow you to define **classes**, **interfaces**, and **methods with a placeholder for types**.
- This provides **type safety** and **code reusability**.
- With Generics, you can write code that works with any data type without casting or duplication
- **Generic Class Syntax:**

```
class ClassName<T> {
    // T is a placeholder for a type
}
```

- **Usage**: When you create an object, you **specify the type**:

```
ClassName<ObjectType> objectName = new ClassName<>();
```

Example – Generic Class

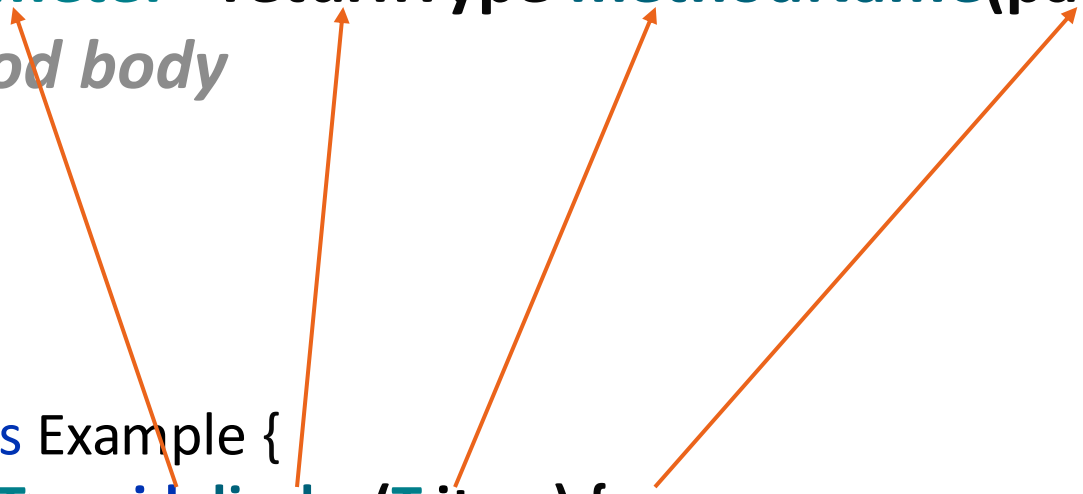
```
class Box<T> {
    private T content;

    // Constructor
    public Box(T content){
        this.content = content;
        System.out.println(content);
    }
    public void setContent(T content) {
        this.content = content;
    }
    public T getContent() {
        return content;
    }
}
public class Main_Box {
        public static void main(String[] args) {
            Box<String> stringBox = new Box<>("Let's Start");
            stringBox.setContent("Hello Generics!");
            System.out.println("String Content: " + stringBox.getContent());

            Box<Integer> intBox = new Box<>(0);
            intBox.setContent(123);
            System.out.println("Integer Content: " + intBox.getContent());
        }
}
```

Output:
String Content: Hello Generics!
Integer Content: 123

Generics (Cont.)

- **General Syntax of a Generic Method:**

```
<typeParameter> returnType methodName(parameterList) {
    // method body
}
```



- Example:
  -
```
public class Example {
    public <T> void display(T item) {
        System.out.println(item);
    }
}
```
  - <T> before return type → means it's a **generic method**.
  - You don't need a generic class to have a generic method.

Printing Array Elements Using Overloaded Method

```
public class GenericMethodTest {
   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.print("Array integerArray contains: ");
      printArray(integerArray); //pass an Integer Array

      System.out.print("Array doubleArray contains: ");
      printArray(doubleArray); //pass a Double Array

      System.out.print("Array characterArray contains: ");
      printArray(characterArray); //pass a Character Array

   }
```

Output:

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

```
// overloaded method printArray
   public static void printArray(Integer[] inputArray)
   {
      // display array elements
      for (Integer element : inputArray)
         System.out.printf("%s ", element);
      System.out.println();
   }
   public static void printArray(Double[] inputArray)
   {
      // display array elements
      for (Double element : inputArray)
         System.out.printf("%s ", element);
      System.out.println();
   }
   public static void printArray(Character[] inputArray)
   {
      // display array elements
      for (Character element : inputArray)
         System.out.printf("%s ", element);
      System.out.println();
   }
} // end class
```

Printing Array Elements Using Generic Method

```
public class GenericMethodTest {
   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.print("Array integerArray contains: ");
      printArray(integerArray); //pass an Integer Array

      System.out.print("Array doubleArray contains: ");
      printArray(doubleArray); //pass a Double Array

      System.out.print("Array characterArray contains: ");
      printArray(characterArray); //pass a Character Array

   }
```

Output:

Array integerArray contains: 1 2 3 4 5
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 method printArray
   public static <T> void printArray(T[] inputArray)
   {
      // display array elements
      for (T element : inputArray)
         System.out.printf("%s ", element);
      System.out.println();
   }
} // end class GenericMethodTest
```

Generic Collections

- **Collections** are **groups of objects** treated as a single unit.
- Java provides a **Collections Framework** — a set of **interfaces**, **classes**, and **algorithms** to store, retrieve, and manipulate groups of data easily.
- **Example**: Instead of managing separate variables, you can manage a list of 100 student names using a List<String>!

Generic Collections (cont.)

Collection Type	Class	Explanation
List	ArrayList<T>	Ordered collection, allows duplicates, dynamic size.
	LinkedList<T>	Like ArrayList but better for frequent inserts/deletes.
Set	HashSet<T>	Unordered, no duplicate elements.
	LinkedHashSet<T>	Ordered version of HashSet (insertion order).
	TreeSet<T>	Sorted set (natural or custom order), no duplicates.
Queue	PriorityQueue<T>	Elements ordered based on priority.
	ArrayDeque<T>	Double-ended queue (add/remove from both ends).
Map	HashMap<K,V>	Key-value pairs, fast lookup, unordered.
	LinkedHashMap<K,V>	Key-value pairs, maintains insertion order.
	TreeMap<K,V>	Sorted key-value pairs based on keys.

Good Programming Practice 20.1

The letters `T` (for "type"), `E` (for "element"), `K` (for "key") and `V` (for "value") are commonly used as type parameters. For other common ones, see `http://docs.oracle.com/javase/tutorial/java/generics/types.html`.

Good Programming Practice 16.1

Avoid reinventing the wheel—rather than building your own data structures, use the interfaces and collections from the Java collections framework, which have been carefully tested and tuned to meet most application requirements.

Generic Collections (cont.)

- **Advantages**:
  - **Dynamic Size**: Collections like ArrayList can grow or shrink at runtime — no need to know size in advance (unlike arrays).
  - **Reusable Algorithms**: Sorting, searching, shuffling — already implemented with utilities like Collections.sort().
  - **Type Safety with Generics**: Generics (<T>) ensure type safety — you avoid casting and runtime errors.
  - **Efficiency**: High-performance implementations are provided for different needs (fast access, quick inserts, sorted data, etc.).
  - **Easy to Maintain and Read Code**: Collections make programs shorter, more understandable, and flexible.
  - **Powerful Data Structures**: Java Collections offer Lists, Sets, Queues, Maps — ready for various real-world needs.

One of the Collection: ArrayList

- It is part of the **Java Collections Framework**.
- It **implements** the **List** interface, which is a type of **Collection**.

Collection (interface)

List (interface)

ArrayList (class)

- ArrayList **is a List**, and
- Every **List** is a type of **Collection**.

One of the Collection: ArrayList (cont.)

Method	Explanation	Example
add(E element)	Adds an element to the end of the list.	names.add("Alice");
add(int index, E element)	Inserts element at specific position.	names.add(1, "Bob");
get(int index)	Returns element at given index.	String name = names.get(0);
set(int index, E element)	Replaces element at index with new one.	names.set(0, "Charlie");
remove(int index)	Removes element at the given index.	names.remove(1);
size()	Returns number of elements in list.	int count = names.size();
isEmpty()	Checks if the list is empty.	names.isEmpty();
clear()	Removes all elements from list.	names.clear();
contains(Object o)	Checks if list contains a specific element.	names.contains("Alice");

E = Element type (like String, Integer, etc.)

One of the Collection: ArrayList (cont.)

```
import java.util.ArrayList;

public class SimpleArrayListExample {
    public static void main(String[] args) {
        // Create an ArrayList to store names
        ArrayList<String> names = new ArrayList<>();

        // Add some names to the list
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");

        // Print all names
        for (String name : names) {
            System.out.println(name);
        }
    }
}
```

Output:
Alice
Bob
Charlie

Questions?