BridgeLabz

Employability Delivered

Java Core Concepts – Generics

Java Core Concepts

BridgeLabz
Employability Delivered

- Java Exceptions
- Java Reflections
- Java Annotations
- *Java Generics*
- *Java Properties*
- Java OpenCSV
- Java JSON using Gson

Java Generics

BridgeLabz
Employability Delivered

Java Generics enables programmers to support Programming Logic and Structure for multiple data types. They can be applied to a Method or a Class –

- **Generic Methods –** Enables programmers to specify with a a single method declaration, to accommodate a set of related methods.
- **Generic Class –** Enables programmers to specify with a single class declaration, to accommodate a set of related types.

Java Generic Methods

Generic method can be declared to handle arguments of different types.

Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately.

UC 1

Given an array of Integer, Double and Character, write a program to print the same

- Create PrintArray class and define toPrint method to print corresponding elements to stdout

PrintArray Sample Code

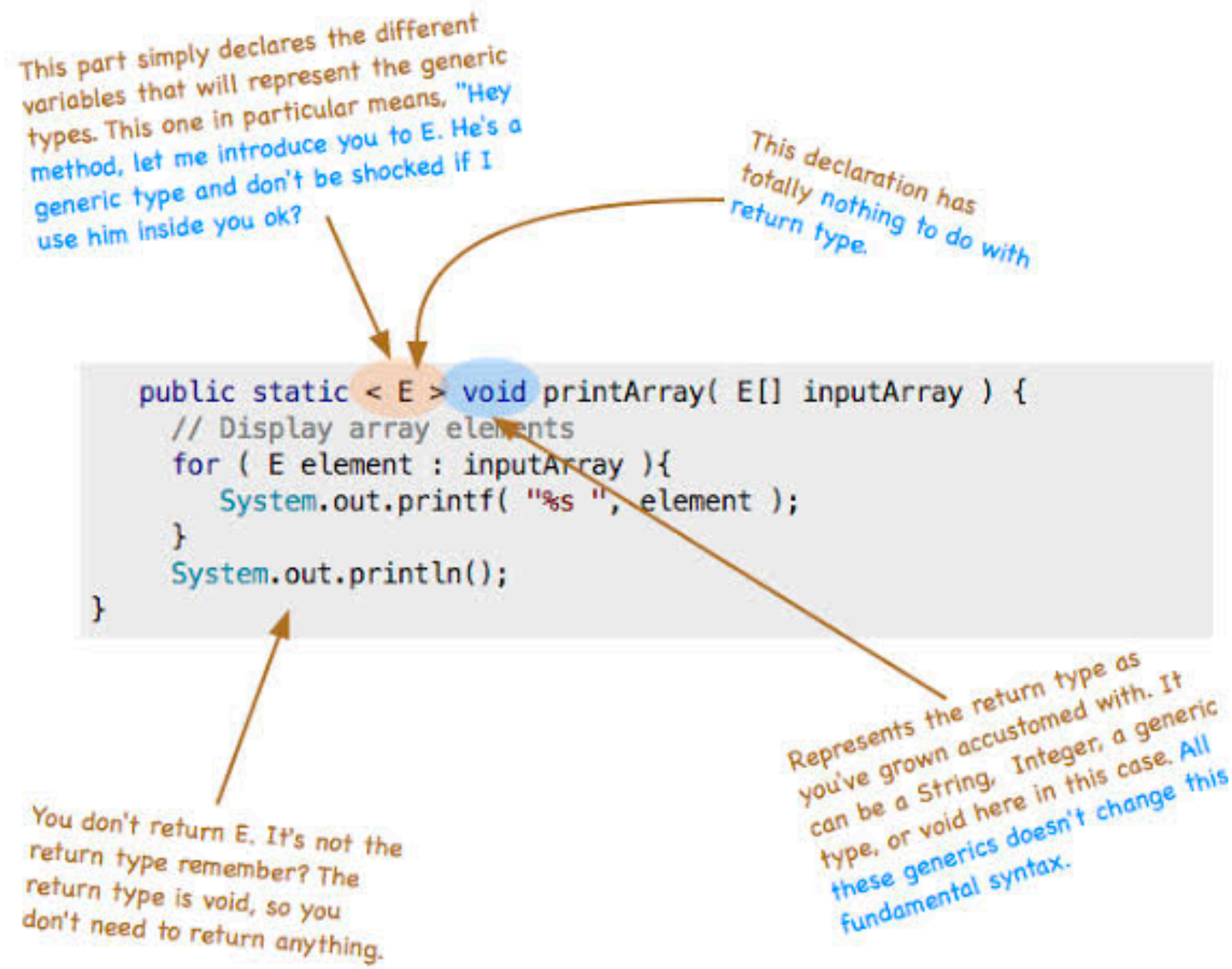
```java
package com.dummyproject;

public class PrintArray {

    public static void toPrint(Integer[] inputArray) {
        for(int element : inputArray) {
            System.out.printf("%s ", element);
        }
        System.out.println();
    }

    public static void toPrint(Double[] inputArray) {
        for(double element : inputArray) {
            System.out.printf("%s ", element);
        }
        System.out.println();
    }

    public static void toPrint(Character[] inputArray) {
        for(char element : inputArray) {
            System.out.printf("%s ", element);
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Integer[] intArray = { 1, 2, 3, 4, 5 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

        PrintArray.toPrint(intArray);
        PrintArray.toPrint(doubleArray);
        PrintArray.toPrint(charArray);
    }
}
```

UC 2

Given an array of Integer, Double and Character, write a program to print the same using Generics

Java Generic Method Declaration

Translated to your problem...



PrintArray Sample Code

```java
package com.dummyproject;

public class PrintArray {

    public static <E> void toPrint(E[] inputArray) {
        for(E element : inputArray) {
            System.out.printf("%s ", element);
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Integer[] intArray = { 1, 2, 3, 4, 5 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

        PrintArray.toPrint(intArray);
        PrintArray.toPrint(doubleArray);
        PrintArray.toPrint(charArray);
    }
}
```

Java Generic Class

BridgeLabz
Employability Delivered

Generic Class can be declared to handle instance variables of different types. To achieve this the class name is followed by a type parameter section.

As with generic methods, the type parameter section of a generic class can have one or more type parameters separated by commas.

These classes are known as parameterized classes or parameterized types because they accept one or more parameters.

Java Generic Method Declaration

Example: Generic Class (Simplified)

```
public class ArrayList<E> {

  public E get(int index) { ... }

  ...
}
```

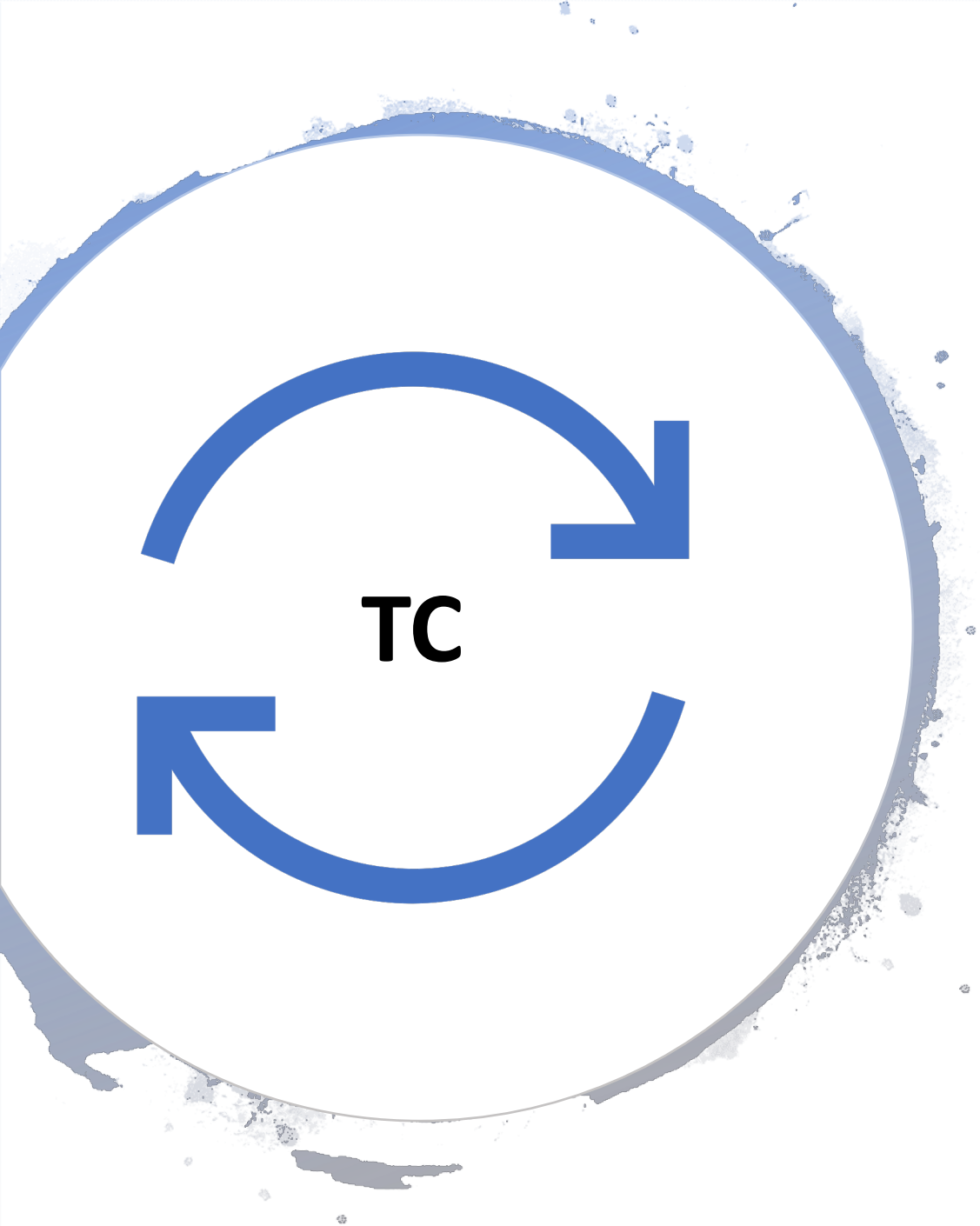
In rest of class, E refers to a type

This says that get returns an E. So, if you created ArrayList<Employee>, get returns an Employee. No typecast required.

This is a highly simplified version of the real java.util.ArrayList class. That class implements multiple interfaces, and the generic support comes from the interfaces.

PrintArray Sample Code

```java
package com.dummyproject;

public class PrintArray<T> {

    private T[] inputArray;

    public PrintArray(T[] inputArray) {
        this.inputArray = inputArray;
    }

    public void toPrint() {
        PrintArray.toPrint(this.inputArray);
    }

    public static <E> void toPrint(E[] inputArray) {
        for(E element : inputArray) {
            System.out.printf("%s ", element);
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Integer[] intArray = { 1, 2, 3, 4, 5 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
        Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

        new PrintArray(intArray).toPrint();
        PrintArray.toPrint(doubleArray);
        PrintArray.toPrint(charArray);
    }
}
```



In the following Use Cases Make sure Test Cases

Test your code is working or not with Test Cases

UC 1

Given 3 Integers find the maximum

- Ensure to test code with the Test Case

UC 2

Given 3 Floats find the maximum

- Ensure to test code with the Test Case

UC 3

Given 3 Strings find the maximum

- Ensure to test code with the Test Case

Refactor 1

Refactor all the 3 to One Generic Method and find the maximum

- Ensure the Generic Type extends Comparable
- Make the test case work

Refactor 2

Refactor to create Generic Class to take in 3 variables of Generic Type

- Ensure the Generic Type extends Comparable
- Write parameter constructor
- Write testMaximum method to internally call the static testMaximum method passing the 3 instance variables
- Define new test case to use the Generic Class

UC 4

Extend the max method to also print the max to std out using Generic Method

- Write printMax Generic Method which is internally called from testMaximum

PrintArray Sample Code

```
public class MaximumTest<T extends Comparable<T>> {

    T x, y, z;

    public MaximumTest(T x, T y, T z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public T maximum() {
        return MaximumTest.maximum(x, y, z);
    }

    // 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 now
        }
        printMax(x, y, z, max);
        return max; // returns the largest object
    }

    public static String testMaximum(String x, String y, String z) {
        String max = x;
        if(y.compareTo(max) > 0) {
            max = y; // y is the largest so far
        }
        if(z.compareTo(max) > 0) {
            max = z; // z is the largest now
        }
        printMax(x, y, z, max);
        return max; // returns the largest object
    }

    public static <T> void printMax(T x, T y, T z, T max) {
        System.out.printf("Max of %s, %s and %s is %s\n",x , y, z, max);
    }

    public static void main(String args[]) {
        Integer xInt = 3, yInt = 4, zInt = 5;
        Float xFl = 6.6f, yFl = 8.8f, zFl = 7.7f;
        String xStr = "pear", yStr = "apple", zStr = "orange";

        MaximumTest.testMaximum(xStr, yStr, zStr);
        new MaximumTest(xInt, yInt, zInt).maximum();
        new MaximumTest(xFl, yFl, zFl).maximum();
        new MaximumTest(xStr, yStr, zStr).maximum();
    }
}
```

BridgeLabz
Employability Delivered

BridgeLabz

Employability Delivered

Thank You