**Assignment Generics**

Q1. What are Generics in Java

**Generics** in Java provide a way to define classes, interfaces, and methods with type parameters, allowing you to write more flexible and reusable code while ensuring type safety at compile time.

**Key Points:**

1. **Type Parameters**: Generics use type parameters (e.g., <T>, <E>, <K, V>) to specify types that will be used by a class, interface, or method. This allows you to define a class or method that can operate on objects of various types while providing compile-time type safety.
2. **Type Safety**: Generics ensure that you can only insert objects of the specified type into a collection or class, preventing ClassCastException at runtime.
3. **Syntax**:
   * **Generic Class**:

public class Box<T> {

private T content;

public void setContent(T content) { this.content = content; }

public T getContent() { return content; }

}

* + **Generic Method**:

public <T> void printArray(T[] array) {

for (T element : array) {

System.out.println(element);

}

}

Q2. What are the benefits of using Generics in Java

Using Generics in Java offers several key benefits:

1. **Type Safety**:
   * **Prevents ClassCastException**: Generics ensure that only the specified type of objects can be added to a collection or used with a class, reducing runtime type errors.
   * **Compile-Time Checking**: Errors related to type mismatch are caught during compilation, not at runtime.
2. **Code Reusability**:
   * **Reusable Code**: Generics allow you to write code that can work with any data type, reducing code duplication. For example, a single generic class can handle multiple types instead of creating multiple classes for each type.
3. **Elimination of Casting**:
   * **Avoids Explicit Casts**: With Generics, you don’t need to cast objects when retrieving them from collections, as the type is known and enforced.
4. **Improved Readability**:
   * **Clearer Code**: Generics provide a way to specify the type of objects a class or method operates on, making code more readable and self-explanatory.
5. **Enhanced Maintainability**:
   * **Easier Maintenance**: Changes related to types are easier to manage because type information is part of the code and checked by the compiler.

Q3. What is a Generic Class in Java

A **Generic Class** in Java is a class that allows you to define the type of objects it can operate on using type parameters. This enables the class to be more flexible and reusable with different data types while ensuring type safety.

**Key Points:**

1. **Type Parameter**:
   * You define a type parameter (e.g., <T>, <E>, <K, V>) that represents the type of objects the class will use.
   * Type parameters are specified within angle brackets <> and used in place of concrete types.
2. **Syntax**:

public class ClassName<T> {

private T field;

public void setField(T field) {

this.field = field;

}

public T getField() {

return field;

}

}

1. **Usage**:
   * **Instantiation**: You instantiate the generic class with specific types when creating objects.

ClassName<String> stringInstance = new ClassName<>();

stringInstance.setField("Hello");

String value = stringInstance.getField(); // No casting needed

* + **With Multiple Type Parameters**:

public class Pair<K, V> {

private K key;

private V value;

public Pair(K key, V value) {

this.key = key;

this.value = value;

}

public K getKey() { return key; }

public V getValue() { return value; }

}

Pair<Integer, String> pair = new Pair<>(1, "One");

**Example:**

Here's an example of a generic class that represents a simple container:public class Box<T> {

private T content;

public void setContent(T content) {

this.content = content;

}

public T getContent() {

return content;

}

}

public class GenericClassExample {

public static void main(String[] args) {

Box<Integer> integerBox = new Box<>();

integerBox.setContent(123);

System.out.println("Integer Box Content: " + integerBox.getContent());

Box<String> stringBox = new Box<>();

stringBox.setContent("Hello Generics");

System.out.println("String Box Content: " + stringBox.getContent());

}

}

Q4.What is a Type Parameter in Java Generics

A **type parameter** in Java Generics is a placeholder for the type of objects that a generic class, interface, or method can operate on. It allows you to write code that can work with any data type while ensuring type safety.

**Key Points:**

1. **Declaration**:
   * Type parameters are defined within angle brackets <> after the class, interface, or method name.
   * Common type parameter names include T (for type), E (for element), K (for key), and V (for value).
2. **Usage**:
   * **Generic Class**:

public class Box<T> {

private T content;

public void setContent(T content) {

this.content = content;

}

public T getContent() {

return content;

}

}

* + **Generic Method**:

public <T> void printArray(T[] array) {

for (T element : array) {

System.out.println(element);

}

}

1. **Instantiation**:
   * When you create an instance of a generic class, you specify the actual type for the type parameter.

Box<String> stringBox = new Box<>();

stringBox.setContent("Hello");

String content = stringBox.getContent();

1. **Type Safety**:
   * Type parameters ensure that only the specified type is used, preventing runtime type errors and the need for type casting.

Q5. What is a Generic Method in Java?

A **Generic Method** in Java is a method that uses type parameters to operate on different types of data while ensuring type safety. It allows you to write methods that can work with various types of inputs without needing multiple method overloads.

**Key Points:**

1. **Type Parameter Declaration**:
   * Type parameters are declared before the return type in the method signature, within angle brackets <>.
2. **Usage**:
   * The method can use the type parameters to specify the types of its parameters and return values.
3. **Example**:

public <T> void printArray(T[] array) {

for (T element : array) {

System.out.println(element);

}

}

* + **Usage**:

Integer[] intArray = {1, 2, 3};

String[] strArray = {"A", "B", "C"};

printArray(intArray); // Prints: 1 2 3

printArray(strArray); // Prints: A B C

Q6. What is the difference between ArrayList and ArrayList<T>

| **List** | **ArrayList** |
| --- | --- |
| List is an interface | ArrayList is a class |
| List interface extends the Collection framework | ArrayList extends AbstractList class and implements List interface |
| List cannot be instantiated. | ArrayList can be instantiated. |
| List interface is used to create a list of elements(objects) that are associated with their index numbers. | ArrayList class is used to create a dynamic array that contains objects. |
| List interface creates a collection of elements that are stored in a sequence and they are identified and accessed using the index. | ArrayList creates an array of objects where the array can grow dynamically. |