Chapter-7: Generics

PART-II: Sample Short Questions & Answers

1. Why do we use Generics in Java?

Generics allow writing code that can work with any data type while providing **type safety** and reducing runtime errors.

👉 Example:

```
ArrayList<String> list = new ArrayList<>();

list.add("Hello");  // Only Strings allowed
```

---

2. Are Generics important in Java?

Yes, Generics are important because they ensure **compile-time type checking**, reusability, and prevent **ClassCastException**.

---

3. When did Java add Generics?

Generics were introduced in **Java 5 (2004)** under the project **Tiger (J2SE 5.0)**.

---

4. What is a Generic type?

A generic type is a class, interface, or method that operates on objects of specified types without being tied to a specific one.

👉 Example:

```
class Box<T> { T data; }
```

---

5. Can we use Generics with Array in Java?

No, we **cannot directly create generic arrays** due to type erasure, but we can use collections like ArrayList<T>.

👉 Example:

```
// Not allowed: T[] arr = new T[5];

ArrayList<Integer> list = new ArrayList<>();
```

---

6. Define classes in generics.

Generic classes allow defining a class with type parameters so it can work with any object type.

👉 Example:

```
class Box<T> {
  T value;
  Box(T v){ value = v; }
}
```

---

7. What are interfaces in generics?

Generic interfaces allow defining interfaces with type parameters that can be specified at implementation.

👉 Example:

```
interface Container<T> { void add(T item); }
```

---

8. Define methods in generics.

Generic methods define type parameters within the method, independent of class-level generics.

👉 Example:

```
public <T> void print(T item){ System.out.println(item); }
```

---

9. Define generic constructor.

A constructor in a generic class can also use generic parameters.

👉 Example:

```
class Data<T> {
  T value;
  <U> Data(U data){ System.out.println(data); }
}
```

---

10. Write about bounded parameters of generics.

Bounded parameters restrict the type argument to a subclass or interface type using extends or super.

👉 Example:

```
class Box<T extends Number> { T data; }
```

PART-III: Sample Long Questions & Answers

1. Briefly describe generics and the uses of generics with example.

- **Definition:** Generics in Java allow defining classes, methods, and interfaces with type parameters.
- **Uses:**
    1. Provides type safety.
    2. Eliminates need for type casting.
    3. Improves code reusability.

    👉 Example:

```
ArrayList<String> list = new ArrayList<>();
list.add("Java");
// String str = list.get(0);  // No casting needed
```

---

2. Describe generic classes and interface with example.

- **Generic Class:** A class with type parameters.

  👉 Example:

```
class Box<T> {
  T value;
  Box(T v){ value = v; }
  T getValue(){ return value; }
}
```

- **Generic Interface:** Interface with type parameter.
  👉 Example:

```
interface Container<T> { void add(T item); }

class MyContainer implements Container<String> {

  public void add(String item){ System.out.println(item); }

}
```

---

3. Describe general type and parameters with example.

- A generic type uses **type parameters** (like <T, U, K, V>) to represent any object type.
  👉 Example:

```
class Pair<K,V> {

  K key; V value;

  Pair(K k, V v){ key=k; value=v; }

}
```

Here K and V are type parameters.

---

4. Explain the use of constructor with generics.

Constructors in generic classes can initialize values of the parameterized type and even use additional type parameters.

👉 Example:

```
class Data<T> {

  T value;

  Data(T v){ value = v; }

  <U> Data(U data){ System.out.println("Extra: " + data); }

}
```

---

5. Define generics, with classes and methods.

- **Generics:** Enables type-safe and reusable code.
- **Generic Class:** Defined with <T>.
- **Generic Method:** Defined with <T> before return type.
  👉 Example:

```
class Box<T> { T item; void set(T i){ item=i; } }


class Demo {

  public <T> void show(T data){ System.out.println(data); }

}
```