Core Java 8 and Development Tools

Lesson 12 : Generics

Capgemini

Lesson Objectives

After completing this lesson, participants will be able to

- Understand concept of Generics
- Implement generic based collections

12.1: Introduction to Generics

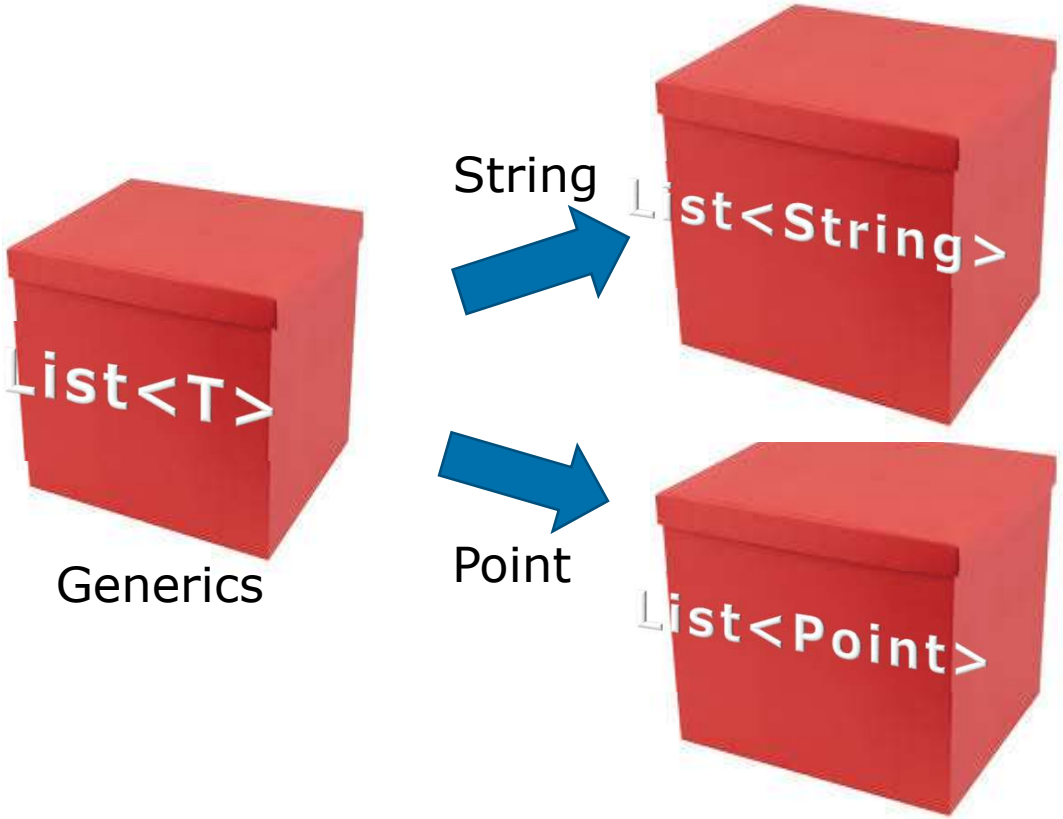
Generics

Generics is a mechanism by which a single piece of code can manipulate many different data types without explicitly having a separate entity for each data type.

12.1: Introduction to Generics
Generics

Generics allows programmer to create parameterized types

Instances of such types can be created by passing reference types



12.2: Writing Generic Classes
Generics Fundamentals

Consider the class given to send the message of type String

Can we reuse the same class to send message of type Employee?

```
public class Sender{
    private String message;
    public setMessage(String message) {
          this.message = message;
    }
    public sendMessage() {
        //logic to send message
     }
}
```

12.2: Writing Generic Classes
Writing Generic Types

How to create a sender class to send generic type of message?

```
public class Sender<T>{
    private T message;
    public setMessage(T message) {
          this.message = message;
    }
    public sendMessage() {
        //logic to send message
     }
}
```

```
Sender<String> stringSender = new Sender<String>();
Sender<Employee> empSender = new Sender<Employee>();
```

12.2: Writing Generic Classes
Generics Terminology

Below listed are different conventions used in generics

Syntax	Meaning
<T>	T denotes instance of any reference type
<?>	? denotes object of any type
<? super T>	? denotes lower bound object of type T
<? extends T>	? denotes upper bound object of type T (class)
<K, V>	K and V denotes instance of any type (same as T)

12.3: Using Generics With Collections
Using Generics with Collections

- Before Generics:

```
List myIntegerList = new LinkedList(); // 1
myIntegerList.add(new Integer(0)); // 2
Integer intObj = (Integer) myIntegerList.iterator().next(); // 3
```

- After Generics:

Note: Line no 3 if not properly typecasted will throw runtime exception

```
List<Integer> myIntegerList = new LinkedList<Integer>(); // 1
myIntegerList.add(new Integer(0)); //2
Integer intObj = myIntegerList.iterator().next(); // 3
```

12.3: Using Generics with Collections
What problems does Generics solve?

Problem: Collection element types:

- Compiler is unable to verify types.
- Assignment must have type casting.
- ClassCastException can occur during runtime.

Solution: Generics

- Tell the compiler type of the collection.
- Let the compiler fill in the cast.
  - **Example:** Compiler will check if you are adding Integer type entry to a String type collection (compile time detection of type mismatch).

12.3: Using Generics with Collections
Using Generic Classes: 1

You can instantiate a generic class to create type specific object.

In J2SE 5.0, all collection classes are rewritten to be generic classes.

- Example:

```
Vector<String> vector = new Vector<String>();
vector.add(new Integer(5)); // Compile error!
vector.add(new String("hello"));
String string = vector.get(0); // No casting needed
```

12.3: Using Generics with Collections
Using Generic Classes: 2

Generic class can have multiple type parameters.

Type argument can be a custom type.

- Example:

```
HashMap<String, Mammal> map =
                                    new HashMap<String, Mammal>();
map.put(“wombat”, new Mammal("wombat"));
Mammal mammal = map.get(“wombat”);
```

12.3: Using Generics with Collections
Generics

Using generics, you can do this:

Object object = new Integer(5);

You can even do this:

Object[] objArr = new Integer[5];

So you would expect to be able to do this: ArrayList<Object> arraylist = new ArrayList<Integer>();

But you can’t do it!!

- This is counter-intuitive at the first glance.

12.3: Using Generics with Collections
Generics

Why does this compile error occur?

- It is because if it is allowed, ClassCastException can occur during runtime – this is not type-safe.

```
ArrayList<Integer> ai = new ArrayList<Integer>();
ArrayList<Object> ao = ai;   // If it is allowed at compile time,
ao.add(new Object());Integer i = ao.get(0);   // will result in runtime
 ClassCastException
```

There is no inheritance relationship between type arguments of a generic class.

12.3: Using Generics with Collections
Generics

The following code works:

```
ArrayList<Integer> ai = new ArrayList<Integer>();
List<Integer> li = new ArrayList<Integer>();
Collection<Integer> ci = new ArrayList<Integer>();
Collection<String> cs = new Vector<String>(4);
```

Inheritance relationship between Generic classes themselves still exists.

12.3: Using Generics with Collections
Generics

The following code works:

```
ArrayList<Number> an = new ArrayList<Number>();
an.add(new Integer(5));
an.add(new Long(1000L));
an.add(new String("hello")); // compile error
```

The entries maintain inheritance relationship.

Summary

Generics

Best practices in Generics

Review Questions

Question 1: If a method created to accept argument of List<Object>, then which of the following are valid options to pass? Ex: void printList(List<Object> list)

- Option1: List<Object>
- Option2: List<Integer>
- Option3: List<Float>
- Option 4: All of the above