Advanced Web Development In Java

Java/J2EE

By Bhuwan Prasad Upadhyay

Advance Java Web Development

Week 2

Part 1 – Advanced Features

- Java Generics and type safety
- Auto-boxing /un-boxing
- Java Enumerations
- Introduction to Enhanced 'For Loop' (for each loop) in java
- Introduction to enhanced if/else
- Java mutable objects – Introduction to StringBuffer and StringBuilder object
- String Tokenizer
- Comparing and identifying objects
- Class/Type casting in java
- Static methods and variables

Java Generics and type safety

- In a nutshell, generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods. Much like the more familiar *formal parameters* used in method declarations, type parameters provide a way for you to re-use the same code with different inputs. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.
- type safety is the extent to which a programming language discourages or prevents type errors. A type error is erroneous or undesirable program behavior caused by a discrepancy between differing data types for the program's constants, variables, and methods (functions), e.g., treating an integer (int) as a floating-point number (float).

Benefits of Generic Code over Non-generic Code

- Code that uses generics has many benefits over non-generic code:
  - Stronger type checks at compile time.
  - Elimination of casts.

```
//without generics requires casting
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0); //Need Cast
```

```
//with generics not requires casting
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);   // no cast
```

  - Enabling programmers to implement generic algorithms.

Keep On Mind About Java Generics

- Generics are implemented using Type Erasure
- Generics does not support sub-typing
- You can't create Generic Arrays
- Use of wildcards with extends or super to increase API flexibility
- Use of Multiple Bounds

Auto-boxing /un-boxing

- Conversion of a primitive type to the corresponding reference type(wrapper class) is called *auto-boxing*.

```
Character ch = 'a';
```

- Conversion of the reference type(wrapper class) to the corresponding primitive type is called *unboxing*.

```
Integer i = new Integer(-8);
//Unboxing through method invocation
int absVal = absoluteValue(i);

public static int absoluteValue(int i) {
    return (i < 0) ? -i : i;
}
```

Java Enumerations

- An *enum type* is a special data type that enables for a variable to be a set of predefined constants.
- In the Java programming language, you define an enum type by using the *enum* keyword. For example, you would specify a days-of-the-week enum type as:

```
public enum Day {

    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY

}
```

- You should use enum types any time you need to represent a fixed set of constants where you know all possible values at compile time.

Introduction to Enhanced 'For Loop' (for each loop) in java

```
List ints = Arrays.asList(1,2,3);
int s = 0;
for (int n : ints) {
    s += n;
}
```

EQUIVALENT

```
for (Iterator it = ints.iterator(); it.hasNext();) {
    int n = it.next();
    s += n;
}
```

Introduction to enhanced if/else

- **"? :" Ternary operator**

```
int x = true ? 10 : 120;
```

x=10

```
int x = false ? 10 : 120;
```

x=120

Java mutable objects – Introduction to StringBuffer and StringBuilder object

- **StringBuilder/StringBuffer** objects are like String objects, except that they can be modified. Internally, these objects are treated like variable-length arrays that contain a sequence of characters. At any point, the length and content of the sequence can be changed through method invocations.
- StringBuilder is not thread safe but StringBuffer is thread safe.
- Both are mutable but String is Immutable class

String Tokenizer

- **StringTokenizer** class allows an application to break a string into tokens.
- This class is a legacy class that is retained for compatibility reasons although its use is discouraged in new code.

Comparing and identifying objects

- Comparing Objects By:
  1. `==` Operator (Reference Comparison)
  2. `equals(Object o)` Method (Value Comparison)
- Identifying Objects :
  - `instanceof` Operator (To identify class type of object)

Class/Type casting in java

- Implicit Casting
  - An implicit cast means you don't have to write code for the cast

```
int i = 10000;
float f = i;
```

- Explicit Casting
  - An explicit cast means you need to write code for the cast

```
float a = 100.001f;
int b = (int)a;
```

Static methods, variables and block

- static variable [*<class-name>.a*]
  - "static" Keyword = Class Variables
  - no "static" Keyword = Instance Variables

```
static int a = 10;
```

- static method [*<class-name>.add(10,20)*]

```
static int add(int a, int b) {
    return a + b;
}
```

- static block [executed when a class is first loaded in to the JVM]

```
static {
    System.out.println("Hello Static Block.");
}
```

Part 2 – Object Collaborations

- Introduction to Object Composition and Class Inheritance
- 'Has a' and 'Is a' relationship
- Java Class Inheritance
- Java Interface and Abstract Classes
- Inner classes
- Java Object Cloning – Shallow and Deep Copy
- Java Serialization
- Polymorphism – Method Overriding and Overloading

Introduction to Object Composition and Class Inheritance

- Composition = Interface

‘Has a’ and ‘Is a’ relationship

Java Interface and Abstract Classes

Inner classes

Java Object Cloning – Shallow and Deep Copy

Java Serialization

Polymorphism – Method Overriding and Overloading

You Can find me:

Twitter : @DeveloperBhuwan

Portfolio : https://devbhuwan.github.io/

My Blog : https://developerbhuwan.blogspot.com/

End Week-2 : Happy Coding.....