# BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 4

# DCA2202

# JAVA PROGRAMMING

# Unit 10

# Other Features in Java

## Table of Contents

| SL No | Topic | | Fig No / Table / Graph | SAQ / Activity | Page No |
|---|---|---|---|---|---|
| 1 | Introduction | | - | - | 3 |
| | 1.1 | Objectives | - | - | |
| 2 | Assertion | | - | 1 | 4 - 5 |
| 3 | Variable Argument (Varargs) | | - | 2 | 6 |
| 4 | Java Static Import | | - | - | 7 |
| 5 | Autoboxing and Unboxing | | 1 | 3 | 8 – 10 |
| 6 | Java Enum | | - | 4 | 11 – 13 |
| 7 | Java Annotation | | - | - | 14 |
| 8 | Java Custom Annotation | | 2, 3 | 5 | 15 - 19 |
| 9 | Summary | | - | - | 19 |
| 10 | Terminal Questions | | - | - | 20 |
| 11 | Answers | | - | - | 21 |
| 12 | Reference | | - | - | 22 |

## 1. INTRODUCTION

In this unit some exciting features of Java has been disused. Assertion feature provides to test the assumption. The "varrags" features allows the method to accept zero or muliple arguments. The static import feature of Java 5 facilitates the java programmer to access any static member of a class directly. Autoboxing and unboxing are introduced in Java 1.5 to automatically convert the primitive type into boxed primitive( Object or Wrapper class).A Java Enum is a special Java type used to define collections of constants.More precisely, a Java enum type is a special kind of Java class. An enum can contain constants, methods etc. Java enums were added in Java 5.Java annotations are used to provide meta data for Java code.

### 1.1 Objectives:

*After studying this unit, you should be able to:*

- *Describe assertion in Java*
- *Explain Variable Argument (Varargs) in Java*
- *Describe the use of Static Import*
- *Explain Autoboxing and Unboxing in Java*
- *Explain Java Enum*
- *Explain Java Annotation*
- *Explain Java Custom Annotation*

## 2. ASSERTION

Assertion is a statement in java. It can be used to test your assumptions about the program.While executing assertion, it is believed to be true. If it fails, JVM will throw an error named **AssertionError**. It is mainly used for testing purpose.The ***assert*** keyword is used in assert statement. Assertion has been introduced in the Java programming language since Java 1.4.

**Syntax of assert statement**

Syntax of an assert statement is as follow (short version):

*assert expression1*;

or (full version):

*assert expression1 : expression2*;

Where:

- expression1 must be a boolean expression.
- expression2 must return a value (must not return void).

The assert statement is working as follows:

- If assertion is enabled, then the assert statement will be evaluated. Otherwise, it does not get executed.
- If expression1 is evaluated to false, an AssertionError error is thrown which causes the program stops immediately. And depending on existence of expression2:
- If expression2 does not exist, then the AssertionError is thrown with no detail error message.
- If expression2 does exist, then a String representation of expression2's return value is used as detail error message.
- If expression1 is evaluate to true, then the program continues normally.

```
package unext.java_examples.assertions;
import java.util.Scanner;

public class AssertionExample {
	public static void main (String args[] ) {
		Scanner scanner = new Scanner( System.in );
		System.out.print("Enter ur age ");
		int value = scanner.nextInt();
		assert value>=18:" Not valid";
		System.out.println("value is "+value);
	}
}
```

If you use assertion, it will not run simply because assertion is disabled by default. To enable the assertion, -**ea** or -**enableassertions** switch of java must be used.

Compile it by: javac AssertionExample.java

Run it by: java -eaAssertionExample

There are some situations where assertion should be avoiding to use. They are:

1. According to Sun Specification, assertion should not be used to check arguments in the public methods because it should result in appropriate runtime exception e.g. IllegalArgumentException, NullPointerException etc.
2. Do not use assertion, if you don't want any error in any situation.

## Self-Assessment Questions - 1

1. If assertion is enabled, then the assert statement will be evaluated. Otherwise, it does not get executed. (True/False)
2. While executing assertion, it is believed to be true. If it fails, JVM will throw an error named __________ .

## 3. VARIABLE ARGUMENT (VARARGS)

The varrags allows the method to accept zero or multiple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many arguments we will have to pass in the method, varargs is the better approach.

**Syntax of varargs:**

The varargs uses ellipsis i.e. three dots after the data type. Syntax is as follows:

*return_typemethod_name(data_type... variableName){}*

```
class VarargsExample1 {

	static void display(String... values) {
		System.out.println("display method invoked ");
	}

	public static void main(String args[]) {
		// zero argument
		display();

		//four arguments
		display("my","name","is","varargs");
	}
}
```

**Output:**

```
display method invoked
display method invoked
```

While using the varargs, you must follow some rules otherwise program code won't compile. The rules are as follows:

- There can be only one variable argument in the method.
- Variable argument (varargs) must be the last argument.

**Self-Assessment Questions - 2**

3. The __________ allows the method to accept zero or multiple arguments.

## 4. JAVA STATIC IMPORT

In order to access static members, it is necessary to qualify references with the class they came from. For example, one must say:

double r = Math.cos(Math.PI * theta);

The static import feature of Java 5 facilitates the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

Instead, the program imports the members, either individually:

import static java.lang.Math.PI;

Once the static members have been imported, they may be used without qualification:

double r = cos(PI * theta);

Example of static import

```
import static java.lang.System.*;

class StaticImportExample {
      public static void main(String args[]) {
            // The class 'out' can be referenced
            // without System as shown below
            out.println("Hello");
      }
}
```

**Output:**

```
Hello
```

## 5. AUTOBOXING AND UNBOXING

The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing. This is the new feature of Java5. So java programmer doesn't need to write the conversion code. For example, converting an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, this is called unboxing.

Here is the simplest example of autoboxing:

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

Consider the following code:

```
    List<Integer> li = new ArrayList<>();
    for (inti = 1; i< 50; i += 2)
         li.add(i);
```

Although you add the int values as primitive types, rather than Integer objects, to li, the code compiles. Because li is a list of Integer objects, not a list of int values. The compiler does not generate an error because it creates an Integer object from i and adds the object to li. Thus, the compiler converts the previous code to the following at runtime:

```
    List<Integer> li = new ArrayList<>();
    for (int i = 1; i< 50; i += 2)
         li.add(Integer.valueOf(i));
```

Converting a primitive value (an int, for example) into an object of the corresponding wrapper class (Integer) is called autoboxing.

The Java compiler applies autoboxing when a primitive value is:

- Passed as a parameter to a method that expects an object of the corresponding wrapper class.
- Assigned to a variable of the corresponding wrapper class.

  Converting an object of a wrapper type (Integer) to its corresponding primitive (int) value is called unboxing. The Java compiler applies unboxing when an object of a wrapper class is:

- Passed as a parameter to a method that expects a value of the corresponding primitive type.
- Assigned to a variable of the corresponding primitive type.

The Unboxing example shows how this works:

```
import java.util.ArrayList;
import java.util.List;

public class Unboxing {
        public static void main(String[] args) {
                Integer i = new Integer(-8);
                // 1. Unboxing through method invocation
                int absVal = absoluteValue(i);
                System.out.println("absolute value of " + i + " = " + absVal);

                List<Double> ld = new ArrayList<>();

                // Π is autoboxed through method invocation.
                ld.add(3.1416);

                // 2. Unboxing through assignment
                double pi = ld.get(0);
                System.out.println("pi = " + pi);
        }

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

**Output:**

```
absolute value of -8 = 8
pi = 3.1416
```

Autoboxing and unboxing lets developers write cleaner code, making it easier to read. The table 9.1 lists the primitive types and their corresponding wrapper classes, which are used by the Java compiler for autoboxing and unboxing:

**Table 9.1: The primitive types and their corresponding wrapper classes**

| Primitive type | Wrapper class |
|---|---|
| boolean | Boolean |
| byte | Byte |
| char | Character |
| float | Float |
| int | Integer |
| long | Long |
| short | Short |
| double | Double |

## Self-Assessment Questions - 3

4. The __________ feature of Java 5 facilitates the java programmer to access any static member of a class directly.
5. The automatic conversion of primitive data types into its equivalent Wrapper type is known as _______ and opposite operation is known as _______ .
6. The Java compiler applies unboxing when an object of a wrapper class is assigned to a variable of the corresponding _______ type.

## 6. JAVA ENUM

Enum in java is a data type that contains fixed set of constants. It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH,

EAST and WEST) etc. The java Enum constants are static and final implicitly. It is available from JDK 1.5.

Java Enums can be thought of as classes that have fixed set of constants. Because they are constants, the names of an Enum type's fields are in uppercase letters.

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*

*}*

**Example of Java Enum**

```
class EnumExample1 {
            public enum Season {
                  WINTER, SPRING, SUMMER, FALL
            }

            public static void main(String[] args) {
                  for (Season s : Season.values())
                        System.out.println(s);

            }
}
```

**Output:**

WINTER

SPRING

SUMMER FALL

**Defining Java Enum**

The enum can be defined within or outside the class because it is similar to a class.

Java enum example: defined outside class

```
enum Season {
      WINTER, SPRING, SUMMER, FALL
}

class EnumExample2 {
      public static void main(String[] args) {
            Season s = Season.WINTER;
            System.out.println(s);
      }
}
```

**Output:**

WINTER

Java enum example: defined inside class

```
class EnumExample3 {
            enum Season {
                  WINTER, SPRING, SUMMER, FALL;
            }// semicolon(;) is optional here

            public static void main(String[] args) {
                  // enum type is required to access WINTER
                  Season s = Season.WINTER;
                  System.out.println(s);
            }
      }
```

**Output**:

WINTER

**Initializing specific values to the enum constants**

The enum constants have initial value that starts from 0, 1, 2, 3 and so on. But we can initialize the specific value to the enum constants by defining fields and constructors. As specified earlier, Enum can have fields, constructors and methods.

Example of specifying initial value to the enum constants

```
class EnumExample4 {
            enum Season {
                  WINTER(5), SPRING(10), SUMMER(15), FALL(20);

                  private int value;

                  private Season(int value) {
                        this.value = value;
                  }
            }

            public static void main(String args[]) {
                  for (Season s : Season.values())
                        System.out.println(s + " " + s.value);

            }
      }
```

**Output:**

```
WINTER 5
SPRING 10
SUMMER 15
FALL 20
```

### Self-Assessment Questions - 4

7. Java ___________ can be thought of as classes that have fixed set of constants.
8. The enum can be defined within or outside the class (True/False)

## 7. JAVA ANNOTATION

Java annotations are used to provide meta data for Java code. Being meta data, Java annotations do not directly affect the execution of the code, although some types of annotations can actually be used for that purpose. Java annotations were added to Java from Java 5.

Annotations have a number of uses, among them:

- Information for the compiler – Annotations can be used by the compiler to detect errors or suppress warnings.
- Compile-time and deployment-time processing – Software tools can process annotation information to generate code, XML files, and so forth.
- Runtime processing – Some annotations are available to be examined at runtime.

## 8. JAVA CUSTOM ANNOTATION

Java Custom annotations or Java User-defined annotations are easy to create and use. The @interface element is used to declare an annotation. For example:

```
@interface MyAnnotation{}
```

Here, MyAnnotation is the custom annotation name.

There are few points that should be remembered by the programmer.

I. Method should not have any throws clauses
II. Method should return one of the following: primitive data types, String, Class, enum or array of these data types.
III. Method should not have any parameter.
IV. We should attach @ just before interface keyword to define annotation.
V. It may assign a default value to the method.

**Types of Annotation**

**There are three types of annotations.**

1. Marker Annotation
2. Single-Value Annotation
3. Multi-Value Annotation

**Marker Annotation**

An annotation that has no method, is called marker annotation. For example:

```
@interface MyAnnotation{}
```

The @Override and @Deprecated are marker annotations.

**Single-Value Annotation**

An annotation that has one method, is called single-value annotation.

For example:

```
@interface MyAnnotation{
    int value();
}
```

We can provide the default value also. For example:

```
@interface MyAnnotation{

        int value() default 0;

}
```

Let's see the code to apply the single value annotation.

@MyAnnotation(value=10)

The value can be anything.

**Multi-Value Annotation**

An annotation that has more than one method, is called Multi-Value annotation. For example:

```
@interface MyAnnotation {

        int value1();

        String value2();

        String value3();

}
```

We can provide the default value also. For example:

```
@interface MyAnnotation {

        int value1() default 1;

        String value2() default "";

        String value3() default "xyz";

}
```

Let's see the code to apply the multi-value annotation.

```
@MyAnnotation(value1=10,value2="ArunKumar",value3="Ghaziabad")
```

**Built-in Annotations used in custom annotations in java**

@Target

@Retention

@Inherited

@Documented

**@Target**

@Target tag is used to specify at which type, the annotation is used. The java.lang.annotation.ElementTypeenum declares many constants to specify the type of element where annotation is to be applied such as TYPE, METHOD, FIELD etc. The constants of ElementTypeenum has been given in the table 9.2:

**Table 9.2: The constants of ElementTypeenum**

| Element Types | Where the annotation can be applied |
|---|---|
| TYPE | class, interface or enumeration |
| FIELD | fields |
| METHOD | methods |
| CONSTRUCTOR | constructors |
| LOCAL_VARIABLE | local variables |
| ANNOTATION_TYPE | annotation type |
| PARAMETER | parameter |

Example to specify annotation for a class

```
@Target({ ElementType.TYPE })

@interface MyAnnotation {

      int value1();

      String value2();

   }
```

Example to specify annotation for a class, methods or fields

```
@Target({ ElementType.TYPE, ElementType.FIELD, ElementType.METHOD })

@interface MyAnnotation {

      int value1();

      String value2();

}
```

***@Retention***

@Retention annotation is used to specify to what level annotation will be available. The table 9.3 lists the availability of RetentionPolicy.

**Table: 9.3: Availability of RetentionPolicy**

| Retention Policy | Availability |
|---|---|
| RetentionPolicy.SOURCE | refers to the source code, discarded during compilation. It will not be available in the compiled class. |
| RetentionPolicy.CLASS | refers to the .class file, available to java compiler butnot to JVM . It is included in the class file. |
| RetentionPolicy.RUNTIME | refers to the runtime, available to java compiler andJVM . |

Example to specify the RetentionPolicy

@Retention(RetentionPolicy.RUNTIME)

```
@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.TYPE)

@interface MyAnnotation {

        int value1();

        String value2();

}
```

***@Inherited***

By default, annotations are not inherited to subclasses. The @Inherited annotation marks the annotation to be inherited to subclasses.

```
@Inherited

@interface

ForEveryone {

}// Now it will be available to subclass also

class Superclass {

}

class Subclass extends Superclass {

}
```

***@Documented***

The @Documented Marks the annotation for inclusion in the documentation.

**Self-Assessment Questions - 5**

9. The _________ element is used to declare an annotation.
10. An annotation that has no method, is called _________ annotation.
11. The _________ annotation marks the annotation to be inherited to subclasses.

## 9. SUMMARY

- Assertion is a statement in java. It can be used to test your assumptions about the program. While executing assertion, it is believed to be true. If it fails, JVM will throw an error named AssertionError.
- The varrags allows the method to accept zero or muliple arguments.
- The static import feature of Java 5 facilitates the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.
- Converting a primitive value (an int, for example) into an object of the corresponding wrapper class (Integer) is called autoboxing. Converting an object of a wrapper type (Integer) to its corresponding primitive (int) value is called unboxing.
- Enum in java is a data type that contains fixed set of constants.
- Java annotations are used to provide meta data for Java code.
- Java Custom annotations or Java User-defined annotations are easy to create and use. The @interface element is used to declare an annotation.

## 10. TERMINAL QUESTIONS

1. What is Autoboxing and Unboxing in Java? Explain with example.
2. What are the different types of annotations? Explain
3. What is the output for the below code? Provide explanation.

```
public class Test {
  public static void main(String[] args) {
          Boolean expr = true;
          if (expr) {
                  System.out.println("true");
          } else {
                  System.out.println("false");
        }
  }
}
```

   Options are

   A) true
   B) Compile Error - can't use Boolean object in if().
   C) false
   D) Compile Properly but Runtime Exception.

4. How to use assertion in Java?
5. Explain Java Static Import with an example.

## 11. ANSWERS

**Self-Assessment Questions**

1. True
2. AssertionError
3. varrags
4. static import
5. boxing,unboxing
6. primitive
7. Enums
8. True
9. @interface
10. marker
11. @Inherited

**Terminal Questions**

1. Converting a primitive value (an int, for example) into an object of the corresponding wrapper class (Integer) is called autoboxing. Converting an object of a wrapper type (Integer) to its corresponding primitive (int) value is called unboxing. (Refer section 5)
2. There are three types of annotations.1) Marker Annotation, 2) Single- Value Annotation and 3) Multi-Value Annotation. (Refer section 8.1)
3. Correct answer is : A
4. Explanations: In the if statement, condition can be Boolean object in jdk1.5 and jdk1.6.
5. Assertion is a statement in java. It can be used to test your assumptions about the program. While executing assertion, it is believed to be true. (Refer section 2)
6. The static import feature of Java 5 facilitates the java programmer to access any static member of a class directly. There is no need to qualify it by the class name. (Refer section 4)

## 12. REFERENCES

• Schildt, Herbert. Java: The Complete Reference. New York: McGraw-Hill Education, 2018.