Java Boot Camp

Manual for Advanced Java

By

Document Control

Contacts

Name	Company	Email	Remark
Sunket Ingale	For AgileTestingAlliance.org		

Change Control

Version	Date	Author(s)	Comments
1.0	23 November 2017	SI	Initial version

Table of Content

1 ABOUT JAVA BOOT CAMP AND THIS MANUAL

Boot Camp

Java Boot Camp is a part of the larger initiative by Agile Testing Alliance known as **#TesterBhiCoder**.

Agile Testing Alliance has been in favor of building a testing community which is more aware about agile and testing. Today's testing world is changing and the demand for technical testers is far more than anyone else.

ATA wants that all the professionals associated with testing and QA step up and learn coding, be more technical than what they are and start pursuing the next generation role, hence the name **#TesterBhiCoder**

ATA is glad that it is able to help fulfil this objective to a large extent. Most of the folks who have registered for the program are into testing for some time and are eager to move to automation specially selenium.

This learning manual is intended for all the attendees of the Java Boot Camp. In an online session this manual will hopefully help them win over Java.

2 OBJECTS AND CONSTRUCTORS IN JAVA

> OBJECTS

Object is an instance of a class. It’s a combination of methods and variables. When we create an object of a class, that object will obtain all the properties of that class (Methods, variables, constructors, etc.).

Eg:

```
MyClass obj = new MyClass(); obj.method1();
```

Where,

a. MyClass is the name of the class

b. obj is the name of the object. Using obj, method1() can be called.

> CONSTRUCTORS

Constructor is a code block which is called and executed at the time of object creation and constructs the values of the object. Constructors look like a method but have the same name as the class which distinguishes them from methods.

Example:

```
public class sample1 {  public static void
        main(String[] args)
        {  sample1 var = new sample1();
        }

        public sample1(){
                System.out.println("Hello World");  }   }
```

Output: Hello World

Note: Multiple objects can be created hence calling the constructor multiple times.

3 INHERITANCE

Inheritance as the name suggests inherits properties from a class (parent) like methods, variables. Only thing is the methods/variables should be non-private. The class which inherits the properties is the child class. We use the 'extends' keyword to inherit child class from parent class.

```
public class Parent {
        String str1 = "Sample1";  String str2
        = "Sample2";  public static String
        str3 = "Sample3";  public static void
        main(String[] args)
        {
                System.out.println("str3 value is "+str3);    }

        public String Method1()
        {  return str1;
        }
}

public class Child extends Parent{ public
        static void main(String[] args)
        {
                Child c = new Child();
                System.out.println("str2 value of parent class is = "+c.str2);
System.out.println("Parent class Method value is "+c.Method1()); } }
```

4 METHOD OVERLOADING

The concept of method overloading is that multiple methods can have the same name but due to their different functionalities can be distinguished by the JVM. The methods can be distinguished by having different number of arguments or by specifying different datatypes for the arguments.

Eg:

```
public static void main(String[] args){
        method1(“Hello”,”World”);
}

public static void method1(String a){
        System.out.println(“Value of variable is: ” + a);
}

             public static void method1(String a,String b){
        System.out.println(“Value of the variables are: ” + a + “ and ” + b);
}
```

Note1 In the above program, the latter method1 will be called because two arguments were given while calling it.

5 CONSTRUCTOR OVERLOADING

Constructor overloading works the same way as method overloading. One can define multiple constructors without different types of arguments and during runtime, it will be defined which constructor gets called.

```
public class ConstructorOverloading {

        public static void main(String[] args){
          ConstructorOverloading obj = new ConstructorOverloading("John","Jordan");
        }

 public ConstructorOverloading(String Name){   System.out.println("The
name is: "+ Name);
        }

  public ConstructorOverloading(String Name,String Location){
System.out.println("The name is: "+ Name);
                System.out.println("The location is: "+ Location);
        }
}
```

Notes:

1. A constructor can be called from another constructor by using the 'this' keyword.

Example: Referring the above program, the second constructor can call the first by:

```
this("John");
```

2. A Sub Class constructor can call a Super Class constructor by using the 'super' keyword.

6 POLYMORPHISM

Polymorphism is an OOPS concept which states that an object can have multiple forms and types. There are two types of polymorphism: Static and Dynamic. Static Polymorphism is achieved through method overloading which we have already discussed. Dynamic polymorphism is determined at runtime and is done through method overriding.

Example:

```
class Tools{

   public void cut(){

        System.out.println("Tools are used to cut");

   }

}

class Knife extends Tools{

   public void cut (){

   System.out.println("Knives can cut and shape objects");

   }

}

class Test{

   public static void main(String[] args){

   Tools ts=new Knife();

   ts.cut();    // prints Knives can cut and shape objects
```

```
    ts =new Vehicle();

    ts.cut();    // prints Tools are used to cut

    }

}
```

7 ABSTRACT METHODS AND CLASSES

A method is said to be abstract when it does not have any sort of implementations done in it. In other words, an abstract method does not have a body but only the name and the arguments.

Eg: abstract void cut(String tool, double length);

An Abstract class is defined with the use of the 'abstract' keyword and it may or may not include abstract methods. Ideally, it is expected that you create abstract methods in abstract classes. An abstract class cannot be instantiated. It does not make sense to instantiate an abstract class since it may only have methods with no bodies. However, an abstract class can be inherited in some other class.

Example:

```
public abstract class Canvas{
        abstract void paint();
}

public          class   PrimaryColours{
public void paint(){
                System.out.println("Paint using primary colours");
        }
}

public          class   SecondaryColours{
public void paint(){
                System.out.println("Paint using secondary colours");
        }
}
```

Why to use Abstract Classes and Methods?

1. When you want to use the same name for the methods but implement them differently in different classes.

2. To share even private/protected methods to the sub classes

3. When you do not want unrelated classes to access the method implementation.

8 INTERFACES

An interface is similar to a class, more so to an abstract class. Except that an interface can only contain abstract methods. In addition to this, an interface can only store constant variables and cannot be instantiated by other classes or interfaces.

Example:

```
public interface SampleInter {

        void draw();
        void paint();
}
```

Note: The methods in Interfaces do not use the ‘abstract’ keyword because it is given that those methods are abstract in nature.

Interfaces are inherited by classes using the ‘implements’ keyword.

Example:

```
public class A implements B{
        //Class Body
}
```

In this, A is a class and B is an interface.

9 ARRAYLISTS

Arraylists like arrays store multiple values in a single variable, but are dynamic in the way that it stores them. It inherits the AbstractList class and implements the List interface. Arraylist are dynamic in storing values in the way that one can edit values which are already stored in the indices. Arraylist objects are instantiated using ArrayList<DataType> class. Where, DataType is the data type of the variable.

Example:

```
ArrayList<String> New1 = new ArrayList<String>();
```

Note that we do not define any length to the ArrayList as it can be dynamically changed.

To add values in the list,

```
New1.add(“value1”);
New1.add(“value2”);
New1.add(“value3”);
```

```
New1.add("value4");
```

Indices are allocated to the List automatically when values are added to it.

To get all the values in an array list:

```
for(int i=0; i<New1.size();i++) {
        System.out.println(New1.get(i));
}
```

Where New1.size() will give the length of the ArrayList.

10 HASHTABLES

Hashtable is a Java Class which inherits the Dictionary class. In Hashtable, the values are stored in the form of key value pairs. The keys need to be unique hence Hashtables are said to have unique elements. They may have null values as well. Hashtables are also synchronised in nature.

Example:

```
Hashtable<String, String> hsh = new Hashtable<String,String>(); hsh.put("Usrn1",
"Bob");   hsh.put("Usrn2",
"Kevin");

System.out.println(hsh.get("Usrn1"));
```

To print all values:

```
for(String key : hsh.keySet()){
        System.out.println(hsh.get(key));
}
```

11 EXCEPTIONS

Exceptions are error events and its important to handle exceptions during execution of test case so that it doesn't halt the test execution during the run time

Handling Exceptions using try-catch block

If we know that a particular code block can generate exceptions then place that code in try catch block

```
 try
{
        System.out.println("Inside the try block");
        System.out.println("Value is "+array[9]);
        System.out.println("Successful");
}
catch(Exception e)
{
        System.out.println("Exception Is "+e);   }
```

Handling Exceptions using throws keyword

If a method is throwing some exception and we need to handle that exception, throws keyword can be used with that method

```
 private static void Method1()
{   try
{
        Exceptionmethod();
} catch (Exception e) {
        System.out.println(e);
        }
}
private static void Exceptionmethod() throws Exception {
        int array[] = {1,2,3};
        System.out.println("Value is "+array[9]);
```

12 ENCAPSULATION

Encapsulation is an OOPS concept which is a process of wrapping code and data into a single unit. An encapsulated class in java keeps its data members as private. Hence, for other classes to be able to access the data need to use certain getter and setter methods.

Advantages:

1. It gives the user control over the data.
2. One can make the file read-only or write-only by using the getter and setter methods.

Example:

```
Class1  private  int
age;

public void sAge(int newAge)
{
        age = newAge;
}

Class2 encap.sAge(12);
```

Where 'encap' is an object of Class1

13 FINAL KEYWORD

Final as the name suggests is a keyword given to variables, methods and classes when you want to limit their use. So in a way we specify final to restrict functionalities of the variables, methods and classes

Variable: A final variable can only either be initialized when it is defined or in a constructor. Once initialized, a final variable value cannot be changed.

```
final abc = 90;
```

Method: A final method cannot be overridden

```
final void abc(){

}
```

Class: A final class cannot be inherited

```
final class abc{
}
```

14 READ-WRITE TEXT FILES

Refer the below example and read the comments for each line

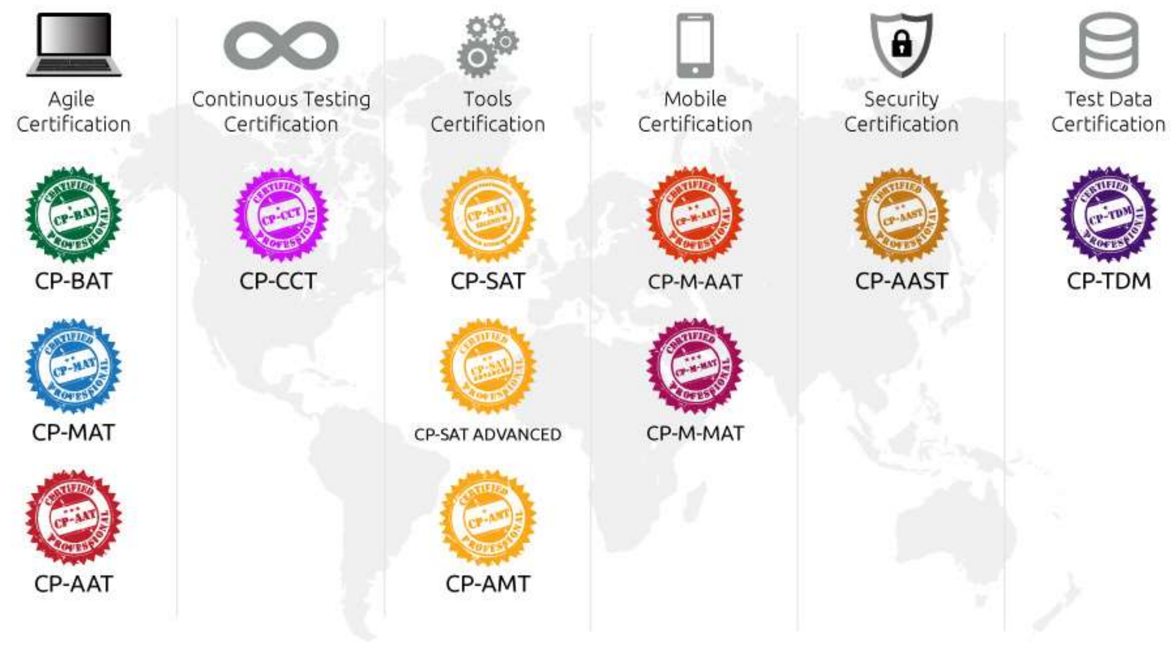
```
public static void main(String[] args) throws IOException
{
        String Test = "D:\\new.txt";   //Specify the file path
        File FC = new File(Test); //Creating a File object and passinf the path.
        FC.createNewFile(); //This would create a new text file
  FileWriter file1 = new FileWriter(Test);//Create object of FileWriter class to edit the file
                  BufferedWriter file2 = new BufferedWriter(file1);//Create object of BufferedWriter class         file2.write("This Is First Line."); //Writing In To File. file2.newLine();//To write next string on new line.             file2.write("This Is Second Line."); //Writing In To File.
        file2.close();

  FileReader file3 = new FileReader(Test); //Create Object of FileReader to read the file
  BufferedReader   file4   =     new   BufferedReader(file3);//Create  Object
        of BufferedReader
        String Content = "sample1"; //To make the variable not null

  //Loop to read all lines one by one from file and print It.
        while((Content = file4.readLine())!= null){
                System.out.println(Content);
        }
        file4.close();
}
```

15 ABOUT ATA

Agile Testing Alliance (ATA) is a non-profit testing community and certification organization, created to grow agile testing awareness, practices and acceptance. ATA is a global alliance of visionary industry leaders, prominent authors, leading educational institutions and testing evangelists who are passionate in proliferation of agile in testing. There is a huge need of agile testing talent and ATA is a step towards filling that void. Our mission is to create a learning roadmap specifically in agile testing space. We understand that learning never stops and that testing community needs recognition in this quest for knowledge.

Hence we have mapped the journey with milestones that can be evaluated, certified and thus recognized.

Here is a snapshot of the learning roadmap



Our Social Media Presence is as below

Website: **http://www.agiletestingalliance.org** **Twitter**
handle @**AgileTA**
Facebook Page: **https://www.facebook.com/AgileTestingAlliance**
Youtube link: https://www.youtube.com/user/AgileTestingAlliance
LinkedIn profile: **http://www.linkedin.com/groups/Agile-Testing-Alliance-5131844**
SlideShare: Learning and sharing Presentations and information **http://www.slideshare.net/AgileTestingAlliance/**
http://www.slideshare.net/ATASlides/