

JAVA



***ADVANCED FEATURES &
PROGRAMMING TECHNIQUES***

NATHAN CLARK

Java

Advanced Features and Programming Techniques

Nathan Clark

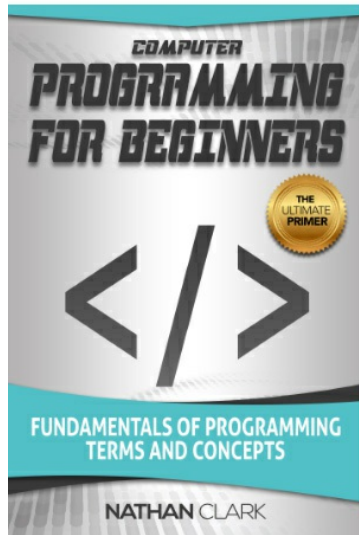
© Copyright 2018 Nathan Clark. All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

Every effort has been made to ensure that the content provided herein is accurate and helpful for our readers at publishing time. However, this is not an exhaustive treatment of the subjects. No liability is assumed for losses or damages due to the information provided.

Any trademarks which are used are done so without consent and any use of the same does not imply consent or permission was gained from the owner. Any trademarks or brands found within are purely used for clarification purposes and no owners are in anyway affiliated with this work.

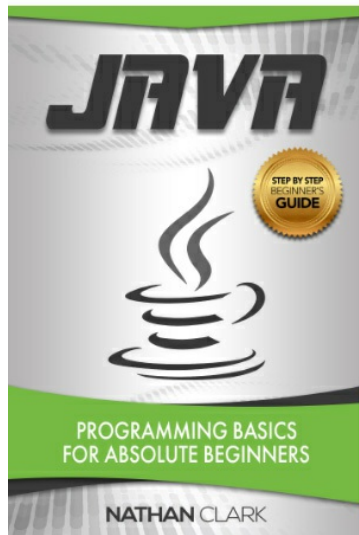
Books in this Series



Computer Programming for Beginners

Fundamentals of Programming Terms and Concepts

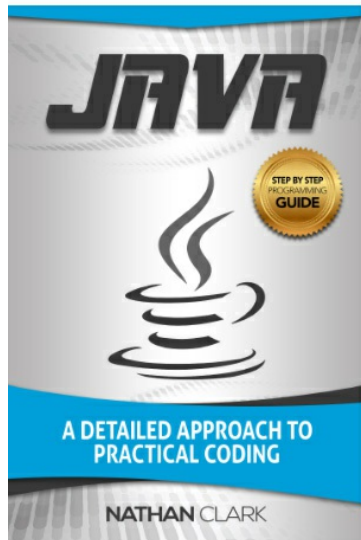
[!\[\]\(e78f798d4ea5c530c9db49e7d26e6b95_img.jpg\) FREE Kindle Version with Paperback](#)



JAVA

Programming Basics for Absolute Beginners

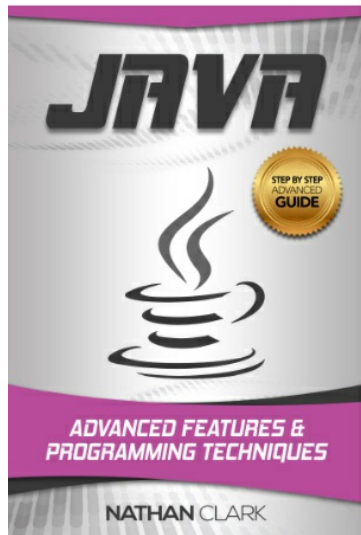
[!\[\]\(05be7c7a8995decd503647c99211f7c2_img.jpg\) FREE Kindle Version with Paperback](#)



JAVA

A Detailed Approach to Practical Coding

 **FREE** Kindle Version with Paperback



JAVA

Advanced Features and Programming Techniques

 **FREE** Kindle Version with Paperback

Table of Contents

[Introduction](#)

[1. Collections](#)

[2. Controlling the App Execution](#)

[3. Packages in Java](#)

[4. Multithreading](#)

[5. Java Annotations](#)

[6. Reflection](#)

[7. Java Properties](#)

[8. Serialization](#)

[9. What is Java EE?](#)

[10. Java Servlets](#)

[11. Java Server Faces](#)

[12. Java Database Connectivity \(JDBC\)](#)

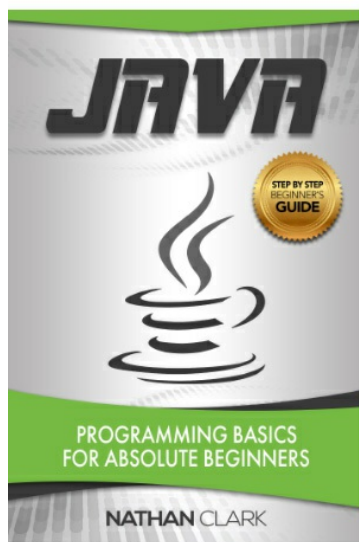
[13. Android and Java](#)

[Conclusion](#)

[About the Author](#)

Introduction

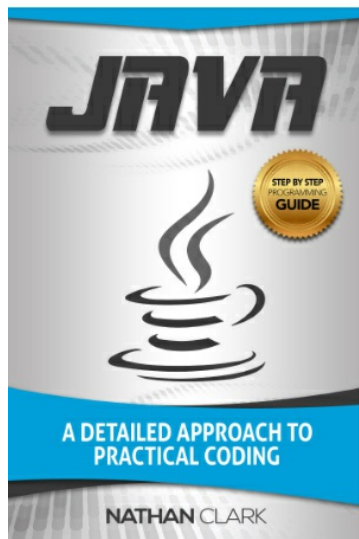
Welcome to the third installment of the Step-By-Step Java series, where I go even further into the workings of Java by looking at advanced features and techniques. If you haven't read the first two books in the series, I highly suggest you do so before getting into this book. Our beginner guide covers the fundamentals of getting started with Java and takes you step by step through writing your very first program, while the second book in the series looks at slightly more complex topics while still being beginner friendly.



JAVA

Programming Basics for Absolute Beginners

 [FREE Kindle Version with Paperback](#)



JAVA

A Detailed Approach to Practical Coding

 [FREE Kindle Version with Paperback](#)

An important aspect of this series, and your learning experience, is **learning**

by doing. Practical examples are proven to be the best way to learn a programming language, which is why I have crammed as many examples into this guide as possible. I have tried to keep the core of the examples similar, so the only variable is the topic under discussion. This makes it easier to understand what we are implementing. As you progress through the chapters, remember to follow along with the examples and try them yourself.

With each topic in this advanced level guide, we will look at a detailed description, proper syntax and numerous examples to make your learning experience as easy as possible. Java is a powerful and versatile programming language and I trust you will enjoy this book as much as I enjoyed writing it.

So without further ado, let's get started!

1. Collections

Collections are special classes in Java that makes working with special data classes much easier. For example, imagine we have a collection called the ArrayList collection, and we want to find out the size of the array. Instead of writing special code to find out the size, we can use a built-in function of the collection called `size()` to get the size of the array.

Some of the notable advantages of collections are:

- It reduces the programming effort required by the developer, since these are built-in available data structures and algorithms that don't have to be written from scratch.
- It increases performance by providing high-performance implementation of data structures and algorithms.
- It provides interoperability between unrelated APIs, by establishing a common language in order to pass collections back and forth.
- It also reduces the effort required to learn APIs, by lessening the need to learn multiple ad hoc collection APIs.

The following collections are present in Java:

- ArrayList - The array container is used to store a contiguous set of values of the same data type.
- HashSet - This collection is used to store elements in such a way that there are no duplicate elements.
- ArrayDeque - This collection is similar to a queue, but here we can add elements at both the beginning and end of the queue.
- HashMap – This is a collection which is a set of key value pairs.

Let's look at each collection in more detail.

1.1 ArrayList

The ArrayList container is used to store a contiguous set of values of the

same data type. Let's look at the definition of an array container via a sample code.

The syntax for defining an array container is as follows:

```
ArrayList variablename=new ArrayList();
```

In this code, 'variablename' refers to the variable name to be assigned to the array list. To add an element to the ArrayList we use the 'add()' method as shown below.

```
Variablename.add(element)
```

Where 'element' is the value that needs to be added to the ArrayList. Let's now look at an example of how to use the ArrayList collection.

Example 1: The following program is used to showcase the way to use array lists.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayList arr=new ArrayList();
        arr.add(1);
        arr.add(2);
        System.out.println("The element at position 1 of the array is "+ arr.get(1));
    }
}
```

With the above program:

- We are first ensuring that we import the 'java.util' package, because this has all the collection classes.
- Next, we define a new object of the ArrayList class.
- We then use the 'add' method to add integer elements to the array list.
- Lastly, we use the 'get' method to get the value of the array list at a particular index position.

With this program, the output is as follows:

The element at position 1 of the array is 2

We can also store other data types, such as strings, in the array list. Another example is shown below:

Example 2: The following program shows how to use array lists with strings.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayList arr=new ArrayList();
        arr.add("One");
        arr.add("Two");
        System.out.println("The element at position 1 of the array is "+ arr.get(1));
    }
}
```

With this program, the output is as follows:

The element at position 1 of the array is Two

We can also store objects of defined classes in the array list. An example of this is shown below:

Example 3: The next program is used to showcase array lists with classes.

```
import java.util.*;
class Person
{
    public String Name;
}
public class Demo
{
    public static void main(String args[])
    {
        ArrayList arr=new ArrayList();
        Person Per1=new Person();
        Per1.Name="John";
        Person Per2=new Person();
        Per2.Name="Mark";
    }
}
```

```
        arr.add(Per1);
    arr.add(Per2);
        System.out.println("The name of Person One is " + ((Person)arr.get(0)).Name);
    System.out.println("The name of Person Two is " + ((Person)arr.get(1)).Name);
    }
}
```

With the above program:

- We are first defining a class named ‘Person’.
- Then, we create two objects of the ‘Person’ class in the main program.
- We store these objects in the ArrayList.
- Lastly, we display the ‘Person Name’ defined in the ArrayList. Remember that we need to typecast the object to ‘Person’ after retrieving the object from the ArrayList.

With this program, the output is as follows:

The name of Person One is John

The name of Person Two is Mark

Now let’s look at the other methods that are available for the ArrayList class.

1.1.1 Size

This method returns the number of elements in the ArrayList. Let’s see an example of this.

Example 4: The following program is used to showcase how to use the size method.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayList arr=new ArrayList();
        arr.add(1);
        arr.add(2);
        System.out.println("The size of the array is "+arr.size());
    }
}
```

```
}  
}
```

With this program, the output is as follows:

The size of the array is 2

1.1.2 Contains

This method returns True if this array list contains the specified element we are looking for, otherwise it returns False. Let's look at an example of this.

Example 5: The following program showcases the way to use the contains method.

```
import java.util.*;  
public class Demo  
{  
    public static void main(String args[])  
    {  
        ArrayList arr=new ArrayList();  
        arr.add(1);  
        arr.add(2);  
        System.out.println("Does the Array List contain the value 2 "+arr.contains(2));  
    }  
}
```

With this program, the output is as follows:

Does the Array List contain the value 2 true

1.1.3 Remove

This method removes the element at the specified position in the array list. Let's see an example of this.

Example 6: This program is used to showcase the way to use the remove method.

```
import java.util.*;  
public class Demo  
{  
    public static void main(String args[])  
    {
```

```

    ArrayList arr=new ArrayList();
        arr.add(1);
    arr.add(2);
    arr.add(3);
        // We are removing the element at Index no 1
    // Remember that the array List starts from Index No 0
    // Hence the value 2 is stored at Index no 1
    arr.remove(1);
    System.out.println("Does the Array List contain the value 2 "+arr.contains(2));
    }
}

```

With this program, the output is as follows:

Does the Array List contain the value 2 false

1.1.4 Add(int index, E element)

This method inserts a specified element at an identified position in the array list. Let's look at an example of this.

Example 7: The next program is used to showcase the way to use the add at index method.

```

import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayList arr=new ArrayList();
        arr.add(1);
        arr.add(2);
        arr.add(3);
        arr.add(2,4);
        int length=arr.size();
        for(int i=0;i<length;i++)
            System.out.println(arr.get(i));
    }
}

```

With the above program:

- We are using the 'add at index' method to add an element at index number 2 of the array list.

- Then we are using the ‘for loop’ to iterate through the elements of the array list.

With this program, the output is as follows:

1
2
4
3

1.2 HashSet

This collection is used to store elements in such a way that there are no duplicate elements.

The syntax for defining an array container is as follows:

```
HashSet variablename=new HashSet();
```

In this code, ‘variablename’ is the variable name to be assigned to the HashSet. To add an element to the HashSet, we use the ‘add()’ method as shown below.

```
Variablename.add(element)
```

Where ‘element’ again is the value that needs to be added to the HashSet. To iterate through the HashSet collection, we first need to create something known as an ‘Iterator’ object as shown below.

```
Iterator<Integer> itr = variablename.iterator();
```

In this code we need to specify the data type for the items in collection, as well as the data type of the iterator object. The variable name is the collection variable. We can then iterate through the elements using the ‘Next’ method.

```
while (itr.hasNext())
```

Let’s look at an example of how to use the HashSet collection.

Example 8: The following program showcases the hash set collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        Set st = new HashSet();
        st.add(1);
        st.add(2);
        st.add(3);
        Iterator<Integer> itr = st.iterator();
        while (itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

With the above program:

- We are first defining a HashSet object.
- Then we use the ‘add’ method to add objects to the HashSet.
- And lastly, we set an iterator object to iterate through the elements of the HashSet.

With this program, the output is as follows:

1

2

3

Similar to the ArrayList collection, we can also use strings as elements of the HashSet. Below is an example of this.

Example 9: The following program shows the hash set collection with strings as elements.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
```



```
{
    Set st = new HashSet();
        st.add("One");
    st.add("Two");
    st.add("Three");
        Iterator<Integer> itr = st.iterator();
    while (itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

With this program, the output is as follows:

One

Two

Three

Now let's look at the other methods that are available for the HashSet class.

1.2.1 Size

This method returns the number of elements in the HashSet. Let's see an example of this.

Example 10: The next program showcases the size method of the hash set collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        Set st = new HashSet();
        st.add("One");
        st.add("Two");
        st.add("Three");
        System.out.println("The size of the Hash Set is "+st.size());
    }
}
```

With this program, the output is as follows:

The size of the Hash Set is 3

1.2.2 Contains

This method returns True if the HashSet contains the item we specify, else it returns False. Let's see an example of this.

Example 11: The following program showcases the contains method of the hash set collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        Set st = new HashSet();
        st.add("One");
        st.add("Two");
        st.add("Three");
        System.out.println("Does the Hash Set contain the value Two "+st.contains("Two"));
    }
}
```

With this program, the output is as follows:

Does the Hash Set contain the value Two true

1.2.3 Removes

This method removes the specified element from the HashSet if it is present. Let's look at an example of this.

Example 12: This program is used to showcase the removes method of the hash set collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        Set st = new HashSet();
        st.add("One");
        st.add("Two");
        st.add("Three");
```

```
        st.remove("Two");
        System.out.println("Does the Hash Set contain the value Two "+st.contains("Two"));
    }
}
```

With this program, the output is as follows:

Does the Hash Set contain the value Two false

1.3 ArrayDeque

This collection is similar to a queue, but here we can add elements at both the beginning and end of the queue.

The syntax for defining an array container is as follows:

```
ArrayDeque<datatype> variablename=new ArrayDeque<datatype>():
```

Where ‘variablename’ is the variable name to be assigned to the ArrayDeque, and ‘datatype’ is the type of elements that we need to add to the queue. To add an element to the ArrayDeque, we use the ‘add()’ method as shown below.

```
Variablename.add(element)
```

In the above code, ‘element’ refers to the value that needs to be added to the ArrayDeque. To iterate through the ArrayDeque collection, we can use the ‘for each’ statement to iterate through the elements.

Let’s look at an example of how to use the ArrayDeque collection.

Example 13: The following program shows the workings of the ArrayDeque collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayDeque<Integer> arr = new ArrayDeque<Integer>();
        arr.add(1);
        arr.add(2);
        arr.add(3);
        for (Integer i : arr)
```

```
{  
    System.out.println(i);  
}  
}
```

With the above program:

- We are first defining an ArrayDeque object, and we are specifying that we want to store Integer elements in the queue.
- We then use the 'add' method to add objects to the queue.
- Lastly, we use the 'for' statement to display the elements of the queue.

With this program, the output is as follows:

1
2
3

Similar to the ArrayList collection, we can use strings as elements of the ArrayDeque. Below is an example of this.

Example 14: The following program showcases the ArrayDeque collection with strings as elements.

```
import java.util.*;  
public class Demo  
{  
    public static void main(String args[])  
    {  
        ArrayDeque<String> arr = new ArrayDeque<String>();  
        arr.add("One");  
        arr.add("Two");  
        arr.add("Three");  
        for (String str : arr)  
        {  
            System.out.println(str);  
        }  
    }  
}
```

With this program, the output is as follows:

One

Two

Three

Now let's look at the other methods that are available for the `ArrayDeque` class.

1.3.1 Size

This method returns the number of elements in the `HashSet`. Let's see an example of this.

Example 15: The next program shows how to use the size method of the ArrayDeque collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayDeque<String> arr = new ArrayDeque<String>();
        arr.add("One");
        arr.add("Two");
        arr.add("Three");
        System.out.println("The size of the ArrayDeque is "+arr.size());
    }
}
```

With this program, the output is as follows:

The size of the ArrayDeque is 3

1.3.2 Contains

This method returns `True` if the `ArrayDeque` contains a specified item, otherwise it returns `False`. Let's look at an example of this.

Example 16: The following program is used to showcase the contains method of the ArrayDeque collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayDeque<String> arr = new ArrayDeque<String>();
        arr.add("One");
        arr.add("Two");
        arr.add("Three");
        System.out.println("Does the ArrayDeque contain the element "+arr.contains("Two"));
    }
}
```

With this program, the output is as follows:

Does the ArrayDeque contain the element true

1.3.3 Removes

This method removes the specified element from the queue, if it is present. Let's see an example of this.

Example 17: This program showcases the removes method of the ArrayDeque collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayDeque<String> arr = new ArrayDeque<String>();
        arr.add("One");
        arr.add("Two");
        arr.add("Three");
        arr.remove("Two");
        System.out.println("Does the ArrayDeque contain the element "+arr.contains("Two"));
    }
}
```

With this program, the output is as follows:

Does the ArrayDeque contain the element false

1.3.4 addFirst

This method adds an element to the beginning of the queue. Let's see an example of this.

Example 18: The following program shows how to use the addFirst method of the ArrayDeque collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayDeque<String> arr = new ArrayDeque<String>();
        arr.add("One");
        arr.add("Two");
        arr.add("Three");
        arr.addFirst("Zero");
        for (String str : arr)
        {
            System.out.println(str);
        }
    }
}
```

With this program, the output is as follows:

Zero

One

Two

Three

1.3.5 addLast

This method adds an element to the end of the queue. Let's look at an example of this.

Example 19: The next program showcases the addLast method of the ArrayDeque collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
```

```

{
    ArrayDeque<String> arr = new ArrayDeque<String>();
    arr.add("One");
    arr.add("Two");
    arr.add("Three");
    arr.addLast("Four");
    for (String str : arr)
    {
        System.out.println(str);
    }
}
}

```

With this program, the output is as follows:

One

Two

Three

Four

1.3.6 getFirst

This method retrieves the element at the beginning of the queue. Let's see an example of this.

Example 20: The next program is used to show the getFirst method of the ArrayDeque collection.

```

import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayDeque<String> arr = new ArrayDeque<String>();
        arr.add("One");
        arr.add("Two");
        arr.add("Three");
        System.out.println("The element at the beginning of the queue is "+arr.getFirst());
    }
}

```

With this program, the output is as follows:

The element at the beginning of the queue is One

1.3.7 getLast

This method fetches the element at the end of the queue. Let's see an example of this.

Example 21: This program shows how to use the getLast method of the ArrayDeque collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        ArrayDeque<String> arr = new ArrayDeque<String>();
        arr.add("One");
        arr.add("Two");
        arr.add("Three");
        System.out.println("The element at the end of the queue is "+arr.getLast());
    }
}
```

With this program, the output is as follows:

The element at the end of the queue is Three

1.4 HashMap

This collection is used to store a collection of key value pairs. When we retrieve a value, we can do it by specifying the key value.

The syntax for defining an array container is as follows:

```
HashMap variablename=new HashMap();
```

In this code, as shown previously, 'variablename' is the variable name to be assigned to the HashMap. To add an element to the HashMap we use the 'put()' method as shown below.

```
Variablename.put(key,value)
```

In the above, 'key' and 'value' is the key and value pair to be added to the

HashMap. To get the value for a particular key, we can use the ‘get’ method. Let’s look at an example of how to use the HashMap collection.

Example 22: The following program is used to showcase the HashMap collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        HashMap mp = new HashMap();
        mp.put(1,"One");
        mp.put(2,"Two");
        mp.put(3,"Three");
        System.out.println("The value for key value of 2 is "+mp.get(2));
    }
}
```

With the above program we are:

- First defining a HashMap object.
- Then we use the ‘put’ method to add objects to the queue. Here we need to specify key and value pairs for the put method.
- Lastly, we use the ‘get’ method to retrieve the value associated with a particular key.

With this program, the output is as follows:

The value for key value of 2 is Two

Now let’s look at some of the methods available for the HashMap class.

1.4.1 Size

This method displays the number of elements in the HashMap object. Let’s see an example of this.

Example 23: This program is showcases the size method of the HashMap collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        HashMap mp = new HashMap();
        mp.put(1,"One");
        mp.put(2,"Two");
        mp.put(3,"Three");
        System.out.println("The number of elements in the map is "+mp.size());
    }
}
```

With this program, the output is as follows:

The number of elements in the map is 3

1.4.2 containsKey

This method returns True if the map object contains a particular key we specify, else it returns False. Let's see an example of this.

Example 24: The next program is used to show the workings of the containsKey method of the HashMap collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        HashMap mp = new HashMap();
        mp.put(1,"One");
        mp.put(2,"Two");
        mp.put(3,"Three");
        System.out.println("Does the map contain the key 2"+mp.containsKey(2));
    }
}
```

With this program, the output is as follows:

Does the map contain the key 2 true

1.4.3 containsValue

Similar to containsKey, this method returns True if the map object contains a

particular value, rather than key. Let's see an example of this.

Example 25: The following program shows the containsValue method of the HashMap collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        HashMap mp = new HashMap();
        mp.put(1,"One");
        mp.put(2,"Two");
        mp.put(3,"Three");
        System.out.println("Does the map contain the value Two "+mp.containsValue("Two"));
    }
}
```

With this program, the output is as follows:

Does the map contain the value Two true

1.4.4 Removes

This method removes an object based on the key value we specify. Let's look at an example of this.

Example 26: The following program showcases the removes method of the HashMap collection.

```
import java.util.*;
public class Demo
{
    public static void main(String args[])
    {
        HashMap mp = new HashMap();
        mp.put(1,"One");
        mp.put(2,"Two");
        mp.put(3,"Three");
        mp.remove(2);
        System.out.println("Does the map contain the value Two "+mp.containsValue("Two"));
    }
}
```

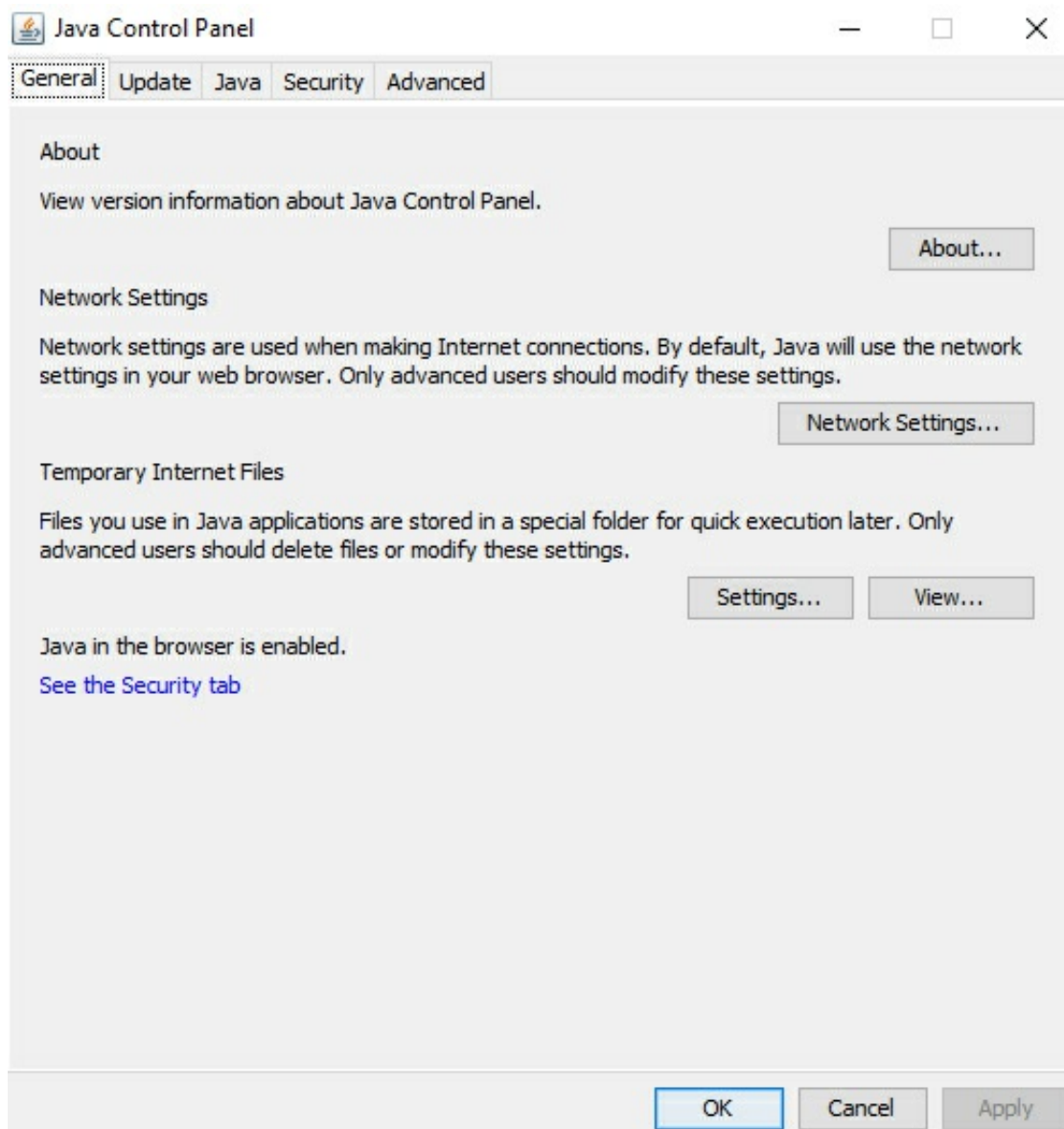
With this program, the output is as follows:

Does the map contain the value Two false

2. Controlling the App Execution

2.1 Security

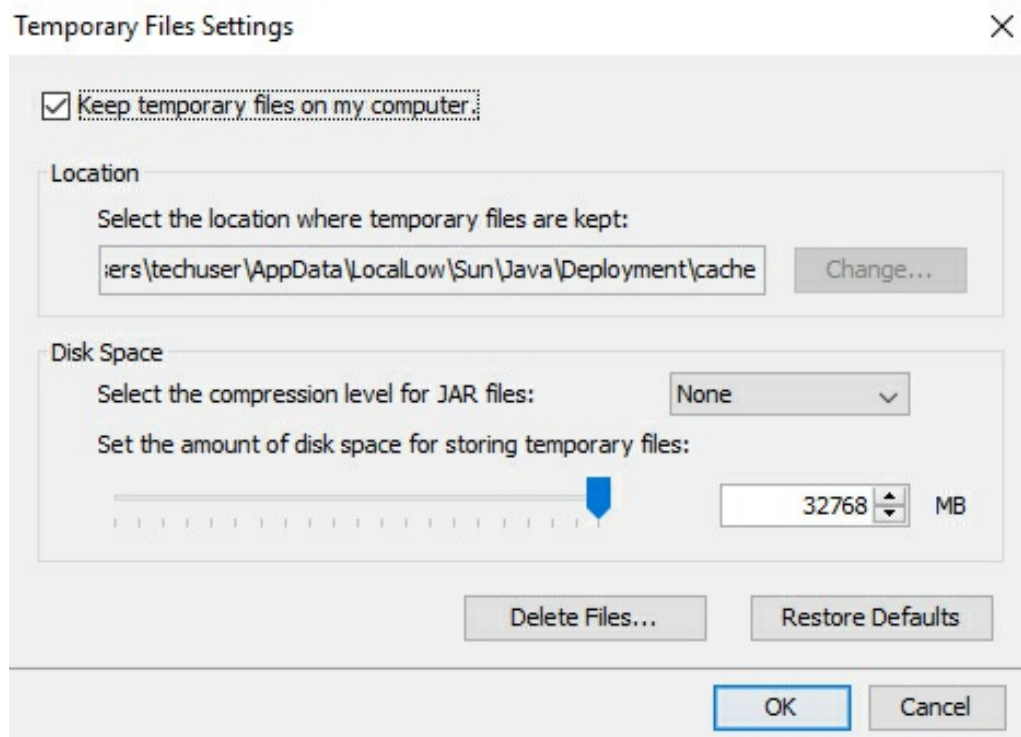
One of the most important aspects when it comes to running programs on workstations is the security that is assigned to the program. Since there are a lot of potentials threats in the industry, it becomes important to control the security aspects of the program. On the Windows platform we have the Java Control Panel, which can be used to control the different aspects of the program.



Let's look at some of the important features of the Java Control Panel.

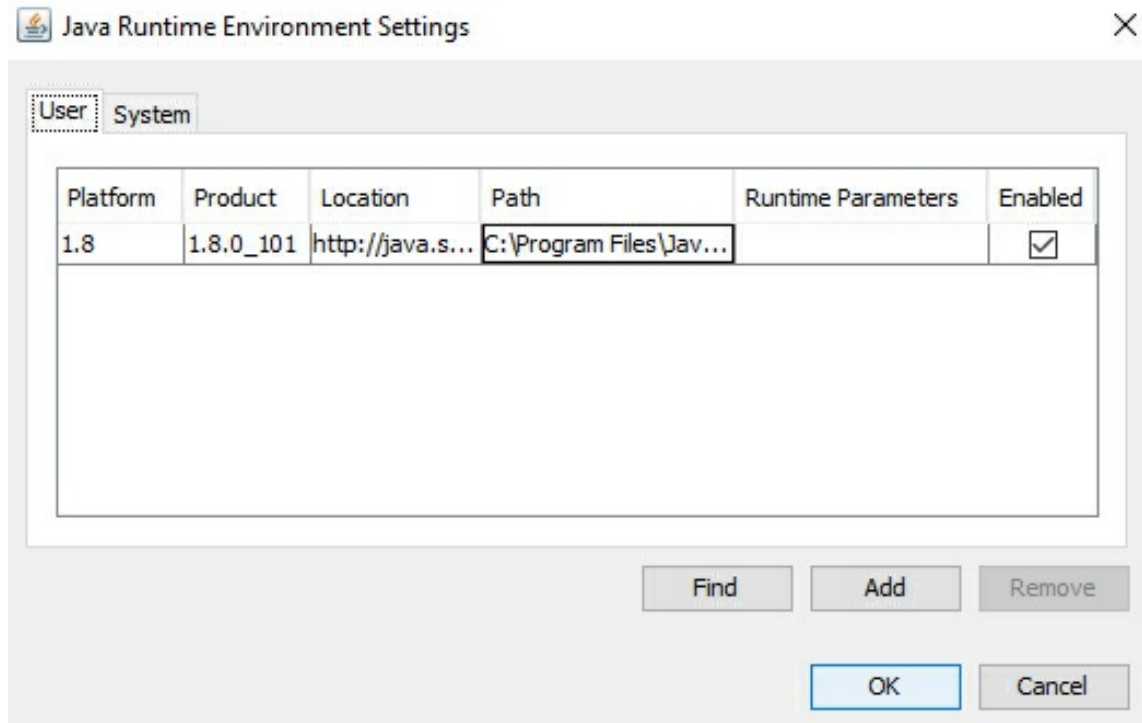
2.1.1 Temp Internet Files

This location is used to store jar files or cache files. It is an important location and you should always ensure that there is sufficient space present for these files.



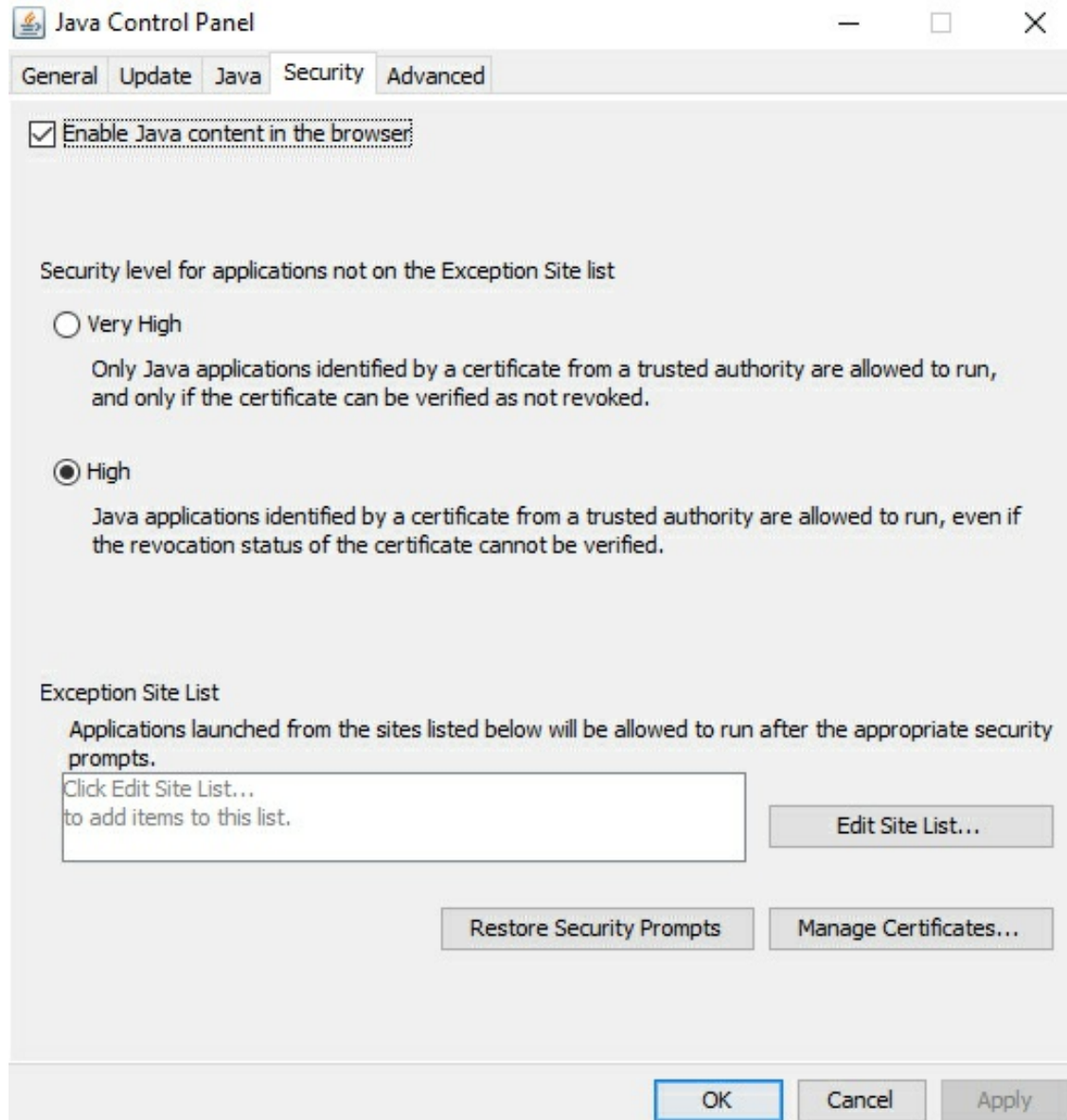
2.1.2 Java Versions

In the Java tab, we can see the various versions of Java that are installed on the system.



2.1.3 Security

In the security tab, we have the various ways in which we can control the security of Java programs running on the system. Here we can disable Java content running in the browser.



2.2 Passing Command Line Arguments

We can also pass command line arguments to control the execution of Java programs. If we look at the main entry point of our Java programs, which is the main method, we will see that we have the String array being passed. Here we can actually pass arguments to the main method. We can then access those values as we would for a normal array.

```
public static void main(String[] args)
```

Let's look at an example of how we can process the arguments in the main

method.

Example 27: The following program shows how to work with arguments in the main method.

```
public class Demo
{
    public static void main(String[] args)
    {
        if(args.length<1)
            System.out.println("No arguments");
        else
        {
            for(String str:args)
                System.out.println(str);
        }
    }
}
```

With the above program:

- We are using the Java command line utility.
- If we store the above program in a file called Demo.java, and then pass the values of 1 and 2 to the program at runtime...

Then we will get the following output:

1

2

3. Packages in Java

Most of the classes and interfaces in Java are organized in what is referred to as 'Packages'. When we worked with collections, we saw that we had to import the java.util package. This is because the collection classes were located in this package.

Java also gives us the ability to create our own packages, organize the class files in these packages, and then access those packages and classes. To ensure that a class is part of a package, we need to define the package statement at the beginning of the class as shown below.

```
package packagename
```

In this code sample, 'packagename' is the name of the package. If we want the class to be accessed outside of the package, then the class needs to be defined as 'public'.

So now let's see how to use packages via an example.

Example 28: The following program showcases the usage of packages in Java.

First we will define a class called Student as shown below:

```
package person;
class Student
{
    public String Name;
    public void Display()
    {
        System.out.println(this.Name);
    }
}
```

The following things need to be noted about the above program:

- First we define the 'Student' class to be in a package named 'person', with a member called 'Name'.
- The class has a method called 'Display', which displays the name

assigned to the object of the class.

Then, let's compile the program as follows:

```
java -d . Student.java
```

Where:

- The '-d' option means that a destination folder, with the same name as the package, should be created.
- The dot(.) indicates that we want to create this in the current folder.

So when we run the above command, we will get a 'Student.class' compiled file that will be in a folder called 'person'.

Now let's create another Java program called 'Demo.java', which will also make use of this packaged class. To do this, let's attach the following code:

```
import person.Student;
class Demo
{
    public static void main(String[] args)
    {
        Student stud =new Student();
        stud.Name="John";
        stud.Display();
    }
}
```

With the above piece of code:

- We can see that we are importing the 'Student' class from the 'person' folder.
- And because the Student class is public, we can make an object and use it like any other class.

With this program, the output is as follows:

John

Note that even if we don't use the import statement, we can use the fully

qualified name of the class as shown below.

```
class Demo
{
    public static void main(String[] args)
    {
        person.Student stud =new person.Student();
        stud.Name="John";
        stud.Display();
    }
}
```

We can also have a nested directory structure for our packages. An example is shown below.

Example 29: This program shows how to use packages in Java with nested directories.

First we will define a class called ‘Student’ as shown below:

```
package com.person;

class Student
{
    public String Name;
    public void Display()
    {
        System.out.println(this.Name);
    }
}
```

Then we compile the program as follows:

```
java -d . Student.java
```

The compiler will create a folder called ‘com’. In that folder it will create a subfolder called ‘person’, and it will place the class file in that folder. Now we need to create the ‘Demo’ class as shown below:

```
import com.person.Student;
class Demo
{
    public static void main(String[] args)
    {
        Student stud =new Student();
        stud.Name="John";
    }
}
```

```
    stud.Display();  
}  
}
```

With this program, the output is as follows:

John

4. Multithreading

Threads are used for concurrent programming. All systems are designed to run programs as threads, which can run concurrently. As a result, programming languages also have the facility to support threaded programming. By running threads, we can run multiple code side by side, giving us more results in a shorter duration of time.

In Java, a thread can be in any one of the following states:

- New - The thread is in the new state when we create an instance of the 'Thread' class, but before the invocation of the 'start()' method.
- Runnable - The thread is in the runnable state after invocation of the 'start()' method.
- Running - The thread is in the running state if the thread scheduler has selected it.
- Non-Runnable (Blocked) - The thread is in this state when the thread is still alive, but is currently not eligible to run.
- Terminated - The thread is terminated or in the dead state when its 'run()' method exits.

All threads are created with the help of either extending the thread class, or the Runnable Interface. Let's look at some examples of threads.

Example 30: The following program shows how to create threads by extending the thread class.

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Thread running");
    }
}

public class Demo
{
    public static void main(String args[])
    {
```

```
MyThread th=new MyThread();
th.start();
}
}
```

The following needs to be noted about the above program:

- First we create a class called ‘MyThread’, which extends the ‘Thread’ class.
- Because we extend the ‘Thread’ class, we need to implement the ‘run’ method.
- In the main program, we create an object of the ‘Thread’ class and choose the ‘start’ method. This will cause our Thread class to call the run method.

With this program, the output is as follows:

Thread running

We can also add more code to the run method, and we can create an array of threads. Let’s look at an example of this.

Example 31: The next program shows how to create an array of threads.

```
class MyThread extends Thread
{
    public static int count=0;
    public void run()
    {
        count++;
        System.out.println("Thread running" + count);
    }
}
public class Demo
{
    public static void main(String args[])
    {
        MyThread[] th=new MyThread[3];
        for(int i=0;i<3;i++)
        {
            th[i]=new MyThread();
            th[i].start();
        }
    }
}
```



```
}

```

The following needs to be noted about the above program:

- In our main thread class, we have a static variable of 'count'.
- We increment the value of count whenever a new thread is created.
- We then create an array of threads in the main program.

With this program, the output is as follows:

Thread running1

Thread running2

Thread running3

Now let's look at an example of how to work with threads using the Runnable Interface.

Example 32: This program shows how to create an array of threads.

```
class MyThread implements Runnable
{
    private Thread t;
    private String threadName;
    MyThread(String name)
    {
        threadName = name;
        System.out.println("Creating " + threadName);
    }
    public static int count = 0;
    public void run()
    {
        count++;
        System.out.println("Thread running" + count);
    }
    public void start()
    {
        System.out.println("Starting " + threadName);
        if (t == null)
        {
            t = new Thread(this, threadName);
            t.start();
        }
    }
}
```

```
}  
}  
public class Demo  
{  
    public static void main(String args[])  
    {  
        MyThread th = new MyThread( "Thread1");  
        th.start();  
    }  
}
```

The following needs to be noted about the above program:

- We are creating our ‘MyThread’ class to implement the Runnable Interface.
- We create a constructor to take a name for the thread and assign it to the ‘private’ variable.
- We also now include a ‘start’ method in which we create a new thread object.

With this program, the output is as follows:

Creating Thread1

Starting Thread1

Thread running1

We can also get the state of the thread at any point in time. This can be done by using the ‘getState’ method on the thread. Let’s see an example of how we can get the state of a thread.

Example 33: The following program is used to showcase how to get the state of a thread.

```
class MyThread extends Thread  
{  
    public static int count=0;  
    public void run()  
    {  
        count++;  
        System.out.println("Thread running" + count);  
    }  
}
```

```
}  
public class Demo  
{  
    public static void main(String args[])  
    {  
        MyThread[] th=new MyThread[3];  
        for(int i=0;i<3;i++)  
        {  
            th[i]=new MyThread();  
            th[i].start();  
            System.out.println(th[i].getState());  
        }  
    }  
}
```

With this program, the output is as follows:

RUNNABLE

RUNNABLE

RUNNABLE

Thread running1

Thread running2

Thread running3

5. Java Annotations

Annotations are used to provide metadata or more information about the program. As such, they don't have any direct effect on the program code itself and are only used to describe the program.

Some examples of how annotations can be used are:

- 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.

The simplest form of an annotation is as follows:

```
@annotationname
```

In the above code line, the '@' symbol indicates to the compiler that this is an annotation. One of the most common annotations you would see is the '@Override' annotation. This is used in the derived class whenever the derived class method overrides the implementation in the base class.

Let's look at an example of this annotation.

Example 34: The following program showcases a simple annotation.

```
class Person
{
    void Display()
    {
        System.out.println("This is the Person class");
    }
}
class Student extends Person
{
    @Override
    void Display()
    {
        System.out.println("This is the Student class");
    }
}
```

```
}  
}  
public class Demo  
{  
    public static void main(String args[])  
    {  
        Person per=new Student();  
        per.Display();  
    }  
}
```

The following things need to be noted about the above program:

- We define 2 classes. One is the ‘Person’ base class and the other is the ‘Student’ derived class.
- Next we define the ‘Display’ method in both classes.
- In the derived class, for the Display method, we add the ‘@Override’ annotation to say that this method will override the definition of the method in the base class. But the annotation itself does not interfere with the program execution.

With this program, the output is as follows:

This is the Student class

5.1 Creating Your Own Annotations

We can also create our own annotations in Java. Let’s look at the steps required to create annotations.

Step 1

First create the annotation type as shown below:

```
@interface annotationname  
{  
    //Annotation members  
}
```

Here the ‘annotationname’ is the name of the annotation. And then we can have members for the annotation. For example, let’s say we want to create an annotation called ‘Author’ and then have a name for the author. We can then

define 'Author' as the name of the annotation, and define a member of the 'String' type with the 'name' value.

Step 2

Append the '@Retention(RetentionPolicy.RUNTIME)' statement to the annotation. This ensures that the annotation is available at runtime.

Step 3

The next step is to access the annotation through reflection. Reflection is the process of gaining access to classes and methods at runtime. There is an entire chapter dedicated to reflection later in this book. But for now, it's good to just know that the concept of getting the annotation values is called reflection.

The best way to understand the entire process is with an example.

Example 35: The following program shows how to create an annotation.

```
import java.lang.annotation.Annotation;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
@Retention(RetentionPolicy.RUNTIME)
@interface Author
{
    String name();
}
@Author(name="John")
class Person
{
    void Display()
    {
        System.out.println("This is the Person class");
    }
}
public class Demo
{
    public static void main(String args[])
    {
        Class<Person> aClass = Person.class;
        Annotation annotation = aClass.getAnnotation(Author.class);
        Author myAnnotation = (Author) annotation;
        System.out.println("name: " + myAnnotation.name());
    }
}
```

The following things need to be noted about the above program:

- We are creating an annotation named 'Author'. This annotation has a member called 'name', which is of the 'String' type.
- We ensure the correct import statements are used, to enable us to work with annotations.
- Next we create a new annotation for the 'Person' class. We then give a name to the 'Author'.
- In the main program, we first get a handle to the 'Person' class.
- We then use this to get the handle to the annotation for the class.
- Finally, we display the value of the annotation.

With this program, the output is as follows:

name: John

We can also apply the annotation to class methods and get the details via reflection. Let's look at an example of this.

Example 36: The next program showcases an annotation for a method.

```
import java.lang.annotation.Annotation;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.reflect.Method;
@Retention(RetentionPolicy.RUNTIME)
@interface Author
{
    String name();
}
class Person
{
    @Author(name="John")
    void Display()
    {
        System.out.println("This is the Person class");
    }
}
public class Demo
{
    public static void main(String args[])
    {
```

```
Class<Person> aClass = Person.class;
Method[] method = aClass.getDeclaredMethods();
Annotation[] annotations = method[0].getDeclaredAnnotations();
for(Annotation annotation : annotations)
{
    Author myAnnotation = (Author) annotation;
    System.out.println("name: " + myAnnotation.name());
}
}
```

The following things need to be noted about the above program:

- We are now applying the annotation for the method.
- We then access the annotation after getting a handler to the method.

With this program, the output is as follows:

name: John

5.2 Predefined Annotation Types

Java has the following list of predefined annotation types:

- **@Deprecated** - This indicates that the marked element is deprecated and should no longer be used. The compiler generates a warning whenever a program uses a method, class or field with the **@Deprecated** annotation.
- **@Override** - This informs the compiler that the element is meant to override another element declared in a superclass.
- **@SuppressWarnings** - This tells the compiler to suppress specific warnings that it would otherwise generate.
- **@SafeVarargs** - When applied to a method or constructor, it asserts that the code does not perform potentially unsafe operations on its varargs parameter.
- **@FunctionalInterface** - Introduced in Java SE 8, it indicates that the type declaration is intended to be a functional interface, as defined by the Java Language Specification.

Let's look at an example using the built-in annotations.

Example 37: This program is used to showcase the @Deprecated annotation.

```
class Person
{
    @Deprecated
    void Display()
    {
        System.out.println("This is the Person class");
    }
}
public class Demo
{
    public static void main(String args[])
    {
        Person per=new Person();
        per.Display();
    }
}
```

Now when we compile the above program, we will get the following compile time error:

Warning:(15, 12) java: Display() in Person has been deprecated

We can still run the program, but we will get the warning every time. This means that the method should be removed from the class in the future. On the other hand, we can also suppress this warning by using the '@SuppressWarnings' annotation. Let's look at an example of this.

Example 38: The following program shows the workings of the @SuppressWarnings annotation.

```
class Person
{
    @Deprecated
    void Display()
    {
        System.out.println("This is the Person class");
    }
}
@SuppressWarnings(value = "deprecation")
```

```
public class Demo
{
    public static void main(String args[])
    {
        Person per=new Person();
        per.Display();
    }
}
```

Now when we compile the program we will not get any warnings, because the '@SuppressWarnings' annotation has been applied to the class that has our main program.

6. Reflection

Reflection is a concept that is present in most modern day programming languages. It allows for the introspection of classes and methods at runtime. So as an example, let's say we don't know what methods are present in a particular class. We can actually inspect the class at runtime, and inspect the methods present in the class. For this to happen we need to use the Reflection API that is present in Java.

Let's look at our first example of how reflection can be used.

Example 39: The following program is an example of reflection.

```
import java.lang.reflect.Method;
class Person
{
}
public class Demo
{
    public static void main(String args[])
    {
        Method[] methods = Person.class.getMethods();
        for(Method method : methods)
        {
            System.out.println("method = " + method.getName());
        }
    }
}
```

The following things need to be noted about the above program:

- We need to import the necessary class from the 'import java.lang.reflect' namespace.
- We are using the '.class' reflection to get the list of methods in the class. This will retrieve all the pre-built methods that come with the Java classes.
- Then for each method, we are printing the name of the methods.

With this program, the output is as follows:

method = wait

method = wait

method = wait

method = equals

method = toString

method = hashCode

method = getClass

method = notify

method = notifyAll

The above implementation retrieves the built-in methods for the class. Let's look at an example of how we can get the declared methods for the class.

Example 40: The following program showcases an example of reflection.

```
import java.lang.reflect.Method;
class Person
{
    public void Display(){}
    public void Input(){}
}
public class Demo
{
    public static void main(String args[])
    {
        Method[] methods = Person.class.getDeclaredMethods();
        for(Method method : methods)
        {
            System.out.println("method = " + method.getName());
        }
    }
}
```

The following needs to be noted about the above program:

- We are using the reflective method called 'getDeclaredMethods' to retrieve the declared methods of the class.

With this program, the output is as follows:

method = Display

method = Input

6.1 Retrieving Information on Classes

Reflection can be used to get information about classes. Let's look at the different aspects of how we can get the information through reflection.

6.1.1 Getting the Class Name

The first aspect is getting the name of the class. Let's look at an example of this.

Example 41: This program shows how to get the name of the class via reflection.

```
import java.lang.reflect.Method;
class Person
{
    public void Display(){}
    public void Input(){}
}
public class Demo
{
    public static void main(String args[])
    {
        Class cls = Person.class;
        System.out.println("The class name is "+cls.getName());
    }
}
```

The following things need to be noted about the above program:

- First we are getting a handle to the class by using class reflection.
- We then use the 'getName' reflective method to get the name of the class.

With this program, the output is as follows:

The class name is Person

6.1.2 Getting the Number of Constructors

We can also get the number of constructors for a class. Let's look at an

example of this.

Example 42: The next program shows how to get the constructors of a class.

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;

class Person
{
    Person()
    {
    }
    public void Display(){}
    public void Input(){}
}
public class Demo
{
    public static void main(String args[])
    {
        Class cls = Person.class;
        Constructor[] constructors = cls.getDeclaredConstructors();
        System.out.println("The number of constructors are " + constructors.length);
    }
}
```

With this program, the output is as follows:

The number of constructors are 1

6.1.3 Getting the Number of Fields

We can get the number of fields for a class. Let's look at an example of this.

Example 43: The following program shows how to get the fields of a class.

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
class Person
{
    int ID;
    Person()
    {
    }
    public void Display(){}
}
```

```
    public void Input(){}  
}  
public class Demo  
{  
    public static void main(String args[])  
    {  
        Class cls = Person.class;  
        Field[] fld = cls.getDeclaredFields();  
        System.out.println("The number of fields are " + fld.length);  
        System.out.println("The fields are");  
        for(Field fldname : fld)  
        {  
            System.out.println(fldname.getName());  
        }  
    }  
}
```

With this program, the output is as follows:

The number of fields are 1

The fields are

ID

6.2 Invoking Methods

We can also invoke methods by the name of the method, after we retrieve the name with reflection. The following steps need to be carried out when working with retrieving method signatures at run time:

Step 1

First we need to create an object of the class through reflection. This is done via the statements below. Here we need to mention the name of the class.

```
Class<?> c = Class.forName(classname);  
Object t = c.newInstance();
```

Step 2

Next we need to get the handle to the method by calling the 'getDeclaredMethod' reflective method.

Step 3

Then we use the ‘Invoke’ function to invoke the method.

Step 4

We also need to ensure that our main method makes an explicit declaration to throw the possible exceptions below, when working with methods and reflection.

- NoSuchMethodException
- InvocationTargetException
- IllegalAccessException
- ClassNotFoundException
- InstantiationException

Now let’s look at an example of how to invoke methods via reflection.

Example 44: The following program shows how to invoke methods via reflection.

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
class Person
{
    int ID;
    Person()
    {
    }
    public void Display()
    {
        System.out.println("This is the Display method");
    }
    public void Input(){}
}
public class Demo
{
    public static void main(String args[]) throws NoSuchMethodException,
    InvocationTargetException, IllegalAccessException, ClassNotFoundException,
    InstantiationException
    {
        Class<?> c = Class.forName("Person");
        Object t = c.newInstance();
        Method meth = Person.class.getDeclaredMethod("Display");
```



```
meth.invoke(t,null);  
}  
}
```

The following things need to be noted about the above program:

- We use the following statements to create a new instance of the 'Person' class at runtime.

```
Class<?> c = Class.forName("Person");  
Object t = c.newInstance();
```

- We then get a handle to the 'Display' method.
- We invoke the method using the newly created object.
- Since there are no parameters for our display method, we need to specify a value of null when invoking the method.

With this program, the output is as follows:

This is the Display method

Let's look at an example of how to invoke a method with parameters.

Example 45: The next program shows how to invoke methods with parameters.

```
import java.lang.reflect.Constructor;  
import java.lang.reflect.Field;  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;  
class Person  
{  
    int ID;  
    public void Display(int pid)  
    {  
        System.out.println("The id is "+pid);  
    }  
    public void Input(){}  
}  
public class Demo  
{  
    public static void main(String args[]) throws NoSuchMethodException,  
    InvocationTargetException, IllegalAccessException, ClassNotFoundException,
```

```
InstantiationException
{
    Class<?> c = Class.forName("Person");
    Object t = c.newInstance();
    Class par1[] = new Class[1];
    par1[0] = Integer.TYPE;
    Object arglist[] = new Object[1];
    arglist[0] = new Integer(10);
    Method meth = Person.class.getDeclaredMethod("Display",par1);
    meth.invoke(t,arglist);
}
}
```

The following things need to be noted about the above program:

- We now have an ‘Integer’ parameter for our ‘Display’ method.
- We create a new parameter type of ‘Integer’ when we call our ‘getDeclaredMethod’ method.
- Then we create a new ‘Integer’ object type and pass in the value of 10.
- When we invoke the method, we pass this value of 10 so that it can be passed over to the method.

With this program, the output is as follows:

The id is 10

6.3 Changing the Values of Fields

We can also change the value of fields assigned to classes via reflection. Let’s look at an example of how we can achieve this.

Example 46: This program shows how to change the value of fields via reflection.

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
class Person
{
    public int ID;
    public String name;
```

```

    public void Display()
    {
        System.out.println("The ID of the person is "+this.ID);
        System.out.println("The name of the person is "+this.name);
    }
    public void Input(){}
}
public class Demo
{
    public static void main(String args[]) throws NoSuchMethodException,
    InvocationTargetException, IllegalAccessException, ClassNotFoundException,
    InstantiationException, NoSuchFieldException
    {
        Class<?> c = Class.forName("Person");
        Object t = c.newInstance();
        Method meth = Person.class.getDeclaredMethod("Display",null);
        Field fldID = c.getField("ID");
        fldID.setInt(t,10);
        Field fldname = c.getField("name");
        fldname.set(t,"John");
        meth.invoke(t,null);
    }
}

```

The following things need to be noted about the above program:

- We add two fields to our class. One being 'ID' and the other being 'name'.
- We first get a handle to the individual fields.
- Then we set the value of the fields with the appropriate 'set' method.

With this program, the output is as follows:

The ID of the person is 10

The name of the person is John

7. Java Properties

Java properties are configuration values that are nothing more than key value pairs. For each pair, the key and the value are both String values. To manage properties, Java provides the 'java.util.Properties' class. This class provides methods for the following:

- Loading key/value pairs into a Properties object from a stream.
- Retrieving a value from its key.
- Listing the keys and their values.
- Enumerating over the keys.
- Saving the properties to a stream.

When we look at some of the use cases for properties, one clear example is the storage of database properties in a file. For example, the properties file can be used to store database values which can then be read from the program.

Let's look at an example on this.

Example 47: The following program showcases how to work with the properties file.

First we create a file called 'db.properties' and we add the following key value pairs to the file:

```
user=system
password=oracle
```

Then we write the following program:

```
import java.io.FileReader;
import java.util.Properties;
public class Demo
{
    public static void main(String args[])
    {
        try
        {
            FileReader reader = new FileReader("H:\\db.properties");
```

```

        Properties pr = new Properties();
pr.load(reader);
        System.out.println("The user name is " + pr.getProperty("user"));
        System.out.println("The password is " + pr.getProperty("password"));
    }
    catch (java.io.FileNotFoundException ex)
    {
        System.out.println(ex.getMessage());
    }
    catch (java.io.IOException IOex)
    {
        System.out.println(IOex.getMessage());
    }
}
}

```

In the above program:

- We are first creating a normal ‘FileReader’ object to read data from the ‘db.properties’ file.
- Then we create a new properties object.
- We load the file object into the properties object.
- And finally, we use the ‘getProperty’ method to retrieve the values of the properties.

With this program, the output is as follows:

The user name is system

The password is oracle

We can also use the ‘Properties’ class to get the system properties. Let’s look at an example of this.

Example 48: The following program shows how to get system properties.

```

import java.util.Iterator;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
public class Demo
{
    public static void main(String args[])

```

```

{
    Properties p = System.getProperties();
    Set set = p.entrySet();
    Iterator itr = set.iterator();
    while (itr.hasNext())
    {
        Map.Entry entry = (Map.Entry) itr.next();
        System.out.println(entry.getKey() + " = " + entry.getValue());
    }
}
}

```

With this program, the output is as follows:

The result returns quite a number of properties. A sample of those properties are shown below.

java.runtime.name = Java(TM) SE Runtime Environment

sun.boot.library.path = C:\Program Files\Java\jdk1.8.0_101\jre\bin

java.vm.version = 25.101-b13

java.vm.vendor = Oracle Corporation

java.vm.name = Java HotSpot(TM) 64-Bit Server VM

We can also create our own properties objects, and utilize them in a program. Let's look at an example of this.

Example 49: The program below shows how to set the properties object.

```

import java.util.Iterator;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
public class Demo
{
    public static void main(String args[])
    {
        Properties p=new Properties();
        p.setProperty("user","system");
        p.setProperty("password","oracle");
        System.out.println("The user name is " + p.getProperty("user"));
        System.out.println("The password is "+ p.getProperty("password"));
    }
}

```

The following need to be noted about the above program:

- We are using the 'setProperty' method to set the name and value of a property.

With this program, the output is as follows:

The user name is system

The password is oracle

We can also write properties to a file. The below code is an example on how this can be achieved.

Example 50: The following program shows how to write properties to a file.

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Iterator;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
public class Demo
{
    public static void main(String args[]) throws IOException
    {
        Properties p=new Properties();
        p.setProperty("user","system");
        p.setProperty("password","oracle");
        System.out.println("The user name is " + p.getProperty("user"));
        System.out.println("The password is "+ p.getProperty("password"));
        p.store(new FileWriter("db.properties"),"Database properties");
    }
}
```

When we run the above program, the code will be written to a file called 'db.properties'.

8. Serialization

Serialization is the concept of storing objects onto persistent storage, such as files or databases. Let's say we have a class known as Person, and during the course of the program we create objects of this class. If we wanted to store the objects so that they could be retrieved later on, we would then use serialization.

In order for the class to be serializable, the class needs to implement the 'Serialize' interface as shown below.

```
public class classname implements Serializable
{
}
```

In this code, 'classname' is the name of the class we want to serialize. We could then use the normal IO streams methods to store the objects to the file system.

Let's look at an example of how we can implement this.

Example 51: This program is used to show how to use serialization.

```
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
class Person implements Serializable
{
    public int ID;
    public String name;
}
public class Demo
{
    public static void main(String args[])
    {
        Person per=new Person();
        per.ID=1;
        per.name="John";
        // Writing the object to the file
        try
        {
            FileOutputStream fout = new FileOutputStream("H:\\Sample.txt");
            ObjectOutputStream out = new ObjectOutputStream(fout);
            out.writeObject(per);
            out.flush();
        }
    }
}
```



```

    }
    catch(java.io.FileNotFoundException ex)
    {
        System.out.println(ex.getMessage());
    }
    catch(java.io.IOException IOex)
    {
        System.out.println(IOex.getMessage());
    }
}
}

```

The following things need to be noted about the above program:

- First we are creating a class called 'Person' with two fields.
- This class implements the 'Serialize' interface.
- We then create a 'FileOutputStream' object in our main program and ensure that the object would be written to a file called 'Sample.txt'.
- Lastly, we use the 'ObjectOutputStream' class to write the object to the file.

When the program is executed, the object would be written in binary format to the Sample.txt file. We can also read the object from the file with a process called deserialization. Let's look at an example of how this is done.

Example 52: The following program shows how to use deserialization.

```

import java.io.*;
class Person implements Serializable
{
    public int ID;
    public String name;
}
public class Demo
{
    public static void main(String args[])
    {
        Person per=new Person();
        per.ID=1;
        per.name="John";
        // Writing the object to the file
    }
}

```

```

try
{
    FileOutputStream fout = new FileOutputStream("H:\\Sample.txt");
    ObjectOutputStream out = new ObjectOutputStream(fout);
    out.writeObject(per);
    out.flush();
    // Now we are deserializing the object
    ObjectInputStream in=new ObjectInputStream(new FileInputStream("H:\\Sample.txt"));
    Person newper=(Person)in.readObject();
    System.out.println("The ID of the Person is "+newper.ID);
    System.out.println("The name of the Person is "+newper.name);
}
catch(java.io.FileNotFoundException ex)
{
    System.out.println(ex.getMessage());
}
catch(java.io.IOException IOex)
{
    System.out.println(IOex.getMessage());
}
catch (ClassNotFoundException e)
{
    e.printStackTrace();
}
}
}

```

The following things need to be noted about the above program:

- We are now adding an ‘InputStream’ object to read the contents of the file.
- Then we use the ‘readObject’ method to read the object from the file.
- We typecast the object returned to the ‘Person’ class.
- And finally we display the ‘ID’ and ‘name’ fields of the object.

With this program, the output is as follows:

The ID of the Person is 1

The name of the Person is John

We can use inherited classes in serialization. If the base class implements the ‘Serialize’ interface, then the objects of the derived class can also be

serialized automatically. Let's look at an example of how we can accomplish this.

Example 53: This program shows how to use base and derived classes in serialization.

```
import java.io.*;
class Person implements Serializable
{
    public int ID;
    public String name;
}
class Student extends Person
{
    public int marks;
}
public class Demo
{
    public static void main(String args[])
    {
        Student st=new Student();
        st.ID=1;
        st.name="John";
        st.marks=10;
        // Writing the object to the file
        try
        {
            FileOutputStream fout = new FileOutputStream("H:\\Sample.txt");
            ObjectOutputStream out = new ObjectOutputStream(fout);
            out.writeObject(st);
            out.flush();
            // Now we are deserializing the object
            ObjectInputStream in=new ObjectInputStream(new FileInputStream("H:\\Sample.txt"));
            Student newst=(Student)in.readObject();
            System.out.println("The ID of the Student is "+newst.ID);
            System.out.println("The name of the Student is "+newst.name);
            System.out.println("The marks of the Student is "+newst.marks);
        }
        catch(java.io.FileNotFoundException ex)
        {
            System.out.println(ex.getMessage());
        }
        catch(java.io.IOException IOex)
        {
            System.out.println(IOex.getMessage());
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }
    }
}
```

```
}  
}  
}
```

With this program, the output is as follows:

The ID of the Student is 1

The name of the Student is John

The marks of the Student is 10

8.1 Transient Keyword

The transient keyword will be used in situations where we don't want a field to be serialized when the entire class is serialized. The transient keyword needs to be used along with the field.

Let's look at an example of how we can accomplish this.

Example 54: The following program showcases the use of the transient keyword.

```
import java.io.*;  
class Person implements Serializable  
{  
    transient int ID;  
    public String name;  
}  
class Student extends Person  
{  
    public int marks;  
}  
public class Demo  
{  
    public static void main(String args[])  
    {  
        Student st=new Student();  
        st.ID=1;  
        st.name="John";  
        st.marks=10;  
        // Writing the object to the file  
        try  
        {  
            FileOutputStream fout = new FileOutputStream("H:\\Sample.txt");  
            ObjectOutputStream out = new ObjectOutputStream(fout);  
            out.writeObject(st);  
        }  
        catch (Exception e)  
        {  
            e.printStackTrace();  
        }  
    }  
}
```

```
out.flush();
    // Now we are deserializing the object
    ObjectInputStream in=new ObjectInputStream(new FileInputStream("H:\\Sample.txt"));
    Student newst=(Student)in.readObject();
    System.out.println("The ID of the Student is "+newst.ID);
    System.out.println("The name of the Student is "+newst.name);
    System.out.println("The marks of the Student is "+newst.marks);
}
catch(java.io.FileNotFoundException ex)
{
    System.out.println(ex.getMessage());
}
catch(java.io.IOException IOex)
{
    System.out.println(IOex.getMessage());
}
catch (ClassNotFoundException e)
{
    e.printStackTrace();
}
}
```

With this program, the output is as shown below. You will notice that the value of 'ID' is defaulting to zero, because this value was not serialized.

The ID of the Student is 0

The name of the Student is John

The marks of the Student is 10

9. What is Java EE?

So far we have looked at how to work with the standard Java programming language. Let's quickly review what the Java world has available for developers and enterprises. There are currently four platforms available:

- Java SE - This is the Standard Edition, which contains all the core Java libraries and the Java API's. Here you also get access to the Java Virtual Machine that is used to run basic Java applications.
- Java ME - This is the Micro Edition of Java. Here you get a subset of features of the main Java Standard Edition. This is good for developing applications on smaller devices such as mobile devices.
- Java FX - This library is used for building rich internet based applications. Here you can design a modern look and feel application with a rich user interface.
- Java EE - Finally you have the Enterprise Edition, which is used for developing enterprise based applications.

Many organizations make use of the Java Enterprise Edition. This edition is based on a set of specifications. For example, the web specification is used for building web based applications. Let's look at some of the components of the Java Enterprise Edition in more detail.

9.1 Java Servlet

Java Servlets are used to build dynamic webpages. The output normally generated by a servlet is HTML. The Java class, that needs to behave like a servlet, needs to conform to the specifications of the Java Servlet. The Java Servlet program is then hosted in a servlet container, which is used to run the servlet and provide the desired output. This is best understood through an example.

Example 55: The following program shows an example of what a servlet looks like.

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DemoServlet extends HttpServlet
{
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        try
        {
            out.println("<html><head>");
            out.println("<title>Hello, World</title></head>");
            out.println("<body>");
            out.println("<h1>Hello, world!</h1>"); // says Hello
            out.println("</body>");
            out.println("</html>");
        }
        finally
        {
            out.close();
        }
    }
}

```

The following things need to be noted about the above program:

- Firstly, the ‘Java’ class extends the ‘HttpServlet’ class. This is required to ensure that the class behaves like a servlet.
- Next, we override the ‘doGet’ method that is used to send the response back to the client.
- And finally, we use the ‘response’ method to write the response back to the client.

This servlet would then run inside a web container. The container would process the servlet and then send the response back to the client. We will look at Java Servlets in greater detail in the next chapter.

9.2 Java Server Pages

Java Server Pages are also used to generate dynamic HTML pages. To

deploy and run Java Server Pages, a compatible web server with a servlet container, such as Apache Tomcat or Jetty, is required. Let's look at an example.

Example 56: The following program shows an example of what a Java server page looks like.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello World Java EE</title>
</head>
<body>
<%System.out.println("Hello World");%>
</body>
</html>
```

The following things need to be noted about the above program:

- The page language directive needs to be declared on top of the program, to indicate that Java constructs can be used in the program.
- The major part of the page is a simple HTML page.
- The '`<% %>`' construct is used to embed Java code.

9.3 Java Server Faces

Java Server Faces (JSF) is used for building component based user interfaces for web based applications. This framework is based on the MVC or the Model/View/Controller framework. Here the Model layer is used to represent the data layer, the View layer is used to denote the presentation layer, and the Controller layer is the business logic layer. Let's look at a small example.

Example 57: The following program shows an example of what a basic JSF program looks like.

A basic Java Server Faces program will contain two parts. The first part is the JSF component itself.

```
import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name="demo")
@SessionScoped
public class Demo implements Serializable
{
    private String str = "Hello World!!";

    public String getstr()
    {
        System.out.println(str);
        return str;
    }
    public void setstr(String pstr)
    {
        this.str = pstr;
    }
}
```

In the above program:

- The '@ManagedBean' annotation indicates the class 'Demo' is a managed bean.
- The '@SessionScoped' indicates that the bean is alive until the 'HttpSession' is valid.
- Then a string 'str' is declared and initialized with 'Hello World' and the getter and setter methods are defined to retrieve the value of string 'str'.

The second part is the xhtml page that references the JSF object.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head>
<title>Example</title>
</h:head>
<h:body>
```

```
        #{demo.str}  
</h:body>  
</html>
```

In the above program, the ‘#{demo.str}’ is used to reference the ‘str’ property from the JSF object. We will look at Java Server Faces in greater detail, later in this book.

10. Java Servlets

As mentioned in the previous chapter, Java Servlets are used to build dynamic web based applications. In order to create a servlet, our class needs to extend one of the main servlet classes. This can be either the 'GenericServlet' class or the 'HttpServlet' class. If the program is going to utilize any protocol, then we need to use the Generic Servlet class, otherwise we use the HttpServlet class.

The basic structure of the Generic Servlet class is shown below.

```
public class classname extends GenericServlet
{
    public void service (ServletRequest request , ServletResponse response)
    throws ServletException , IOException
    {
    }
}
```

The following things need to be noted about the above structure:

- The class has to extend the 'GenericServlet' class
- Then we need to override the service method.
- The service method gets passed the request and the response object. If the user has sent some parameters with the request, it would be available in the request object.

Alternatively if the class was going to work with the HttpServlet class, then the syntax would be as follows:

```
public class Demo extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
    }
}
```

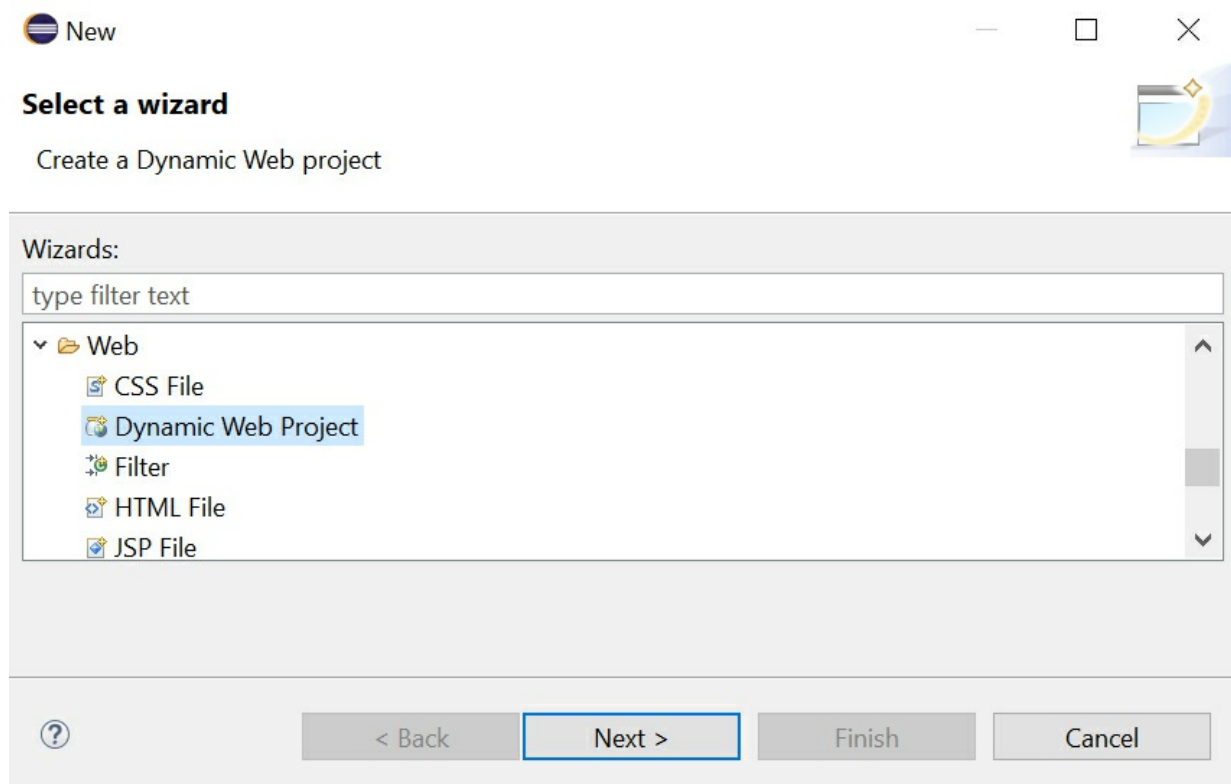
10.1 How to Create a Simple Servlet

To create a simple servlet in an IDE, such as Eclipse, we need to create a

dynamic web project. This dynamic web project will be configured with a web server that will be used to run the servlet program. In Eclipse it is important to ensure that the web tools are installed. The Eclipse web tools can download from the link below:

<https://www.eclipse.org/webtools/>

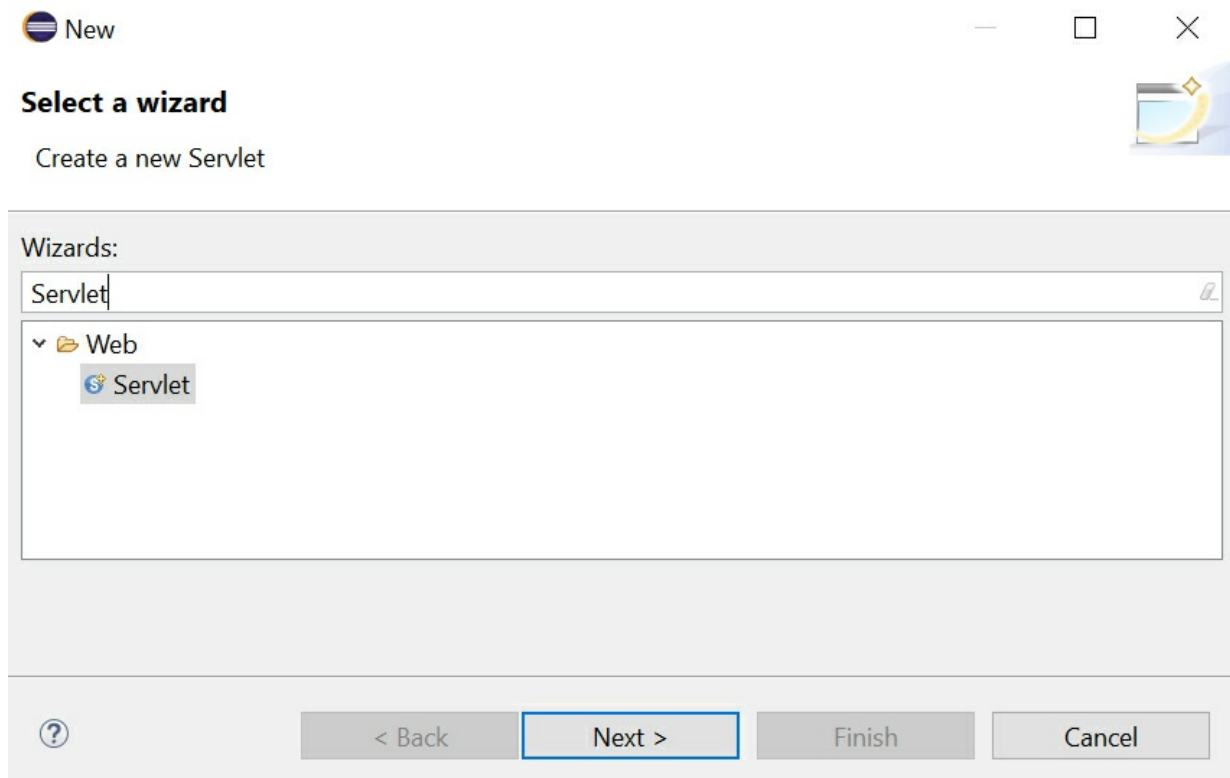
With this installed, when we go to Eclipse and create an 'Other' type project, we will be able to see the 'Dynamic Web Project' under the 'Web' section.



We also need to ensure that a web server, such as Apache is downloaded. Apache can be downloaded from the following link:

<https://httpd.apache.org/>

Then in Eclipse, we need to click on the project and add a new servlet. The following dialog box will be shown shown.



Once the servlet has been created, the skeleton code below will be added for you.

Example 58: The following program shows how to create a basic servlet program.

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Demo
 */
@WebServlet("/Demo")
public class Demo extends HttpServlet
{
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
}
```

```

public Demo()
{
    super();
    // TODO Auto-generated constructor stub
}

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
{
    // TODO Auto-generated method stub
    response.getWriter().append("Hello World");
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
{
    // TODO Auto-generated method stub
    doGet(request, response);
}
}

```

The following things need to be noted about the above program:

- First the class is extending the servlet class.
- It is then overriding the ‘doGet’ and ‘doPost’ methods that come as part of the boiler plate code.
- In the ‘doGet’ method, we then add the following line of code to write ‘Hello World’ as the response.

```
response.getWriter().append("Hello World");
```

When we run the program in Eclipse, it will run in Apache in Eclipse and we will get the following output:

Hello World

10.2 Using Parameters

With dynamic web pages, it is possible to send parameters via the Query

String in the URL. We can then use the Java servlet to process this parameter. Let's see an example of how we can get this done.

Example 59: The following program showcases how to use parameters.

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Demo
 */
@WebServlet("/Demo")
public class Demo extends HttpServlet
{
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Demo()
    {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        // TODO Auto-generated method stub
        String id=request.getParameter("Id");
        String name=request.getParameter("name");

        PrintWriter out=response.getWriter();
        out.println("<p> Id is "+id + "</p>");
        out.println("<p> Name is "+name + "</p>");
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
```

```
{  
    // TODO Auto-generated method stub  
    doGet(request, response);  
}
```

The following things need to be noted about the modified program:

- In the 'doGet' method, we now create a 'PrintWriter' object to write data back as a response.
- We then use the 'request.getParameter' object to get the query string parameters. Here we are assuming that we are passing an 'ID' and 'Name' as the query string parameters. We are then writing these values back in the response.
- Finally, we are embedding these values in html tags and then writing the response back to the client.

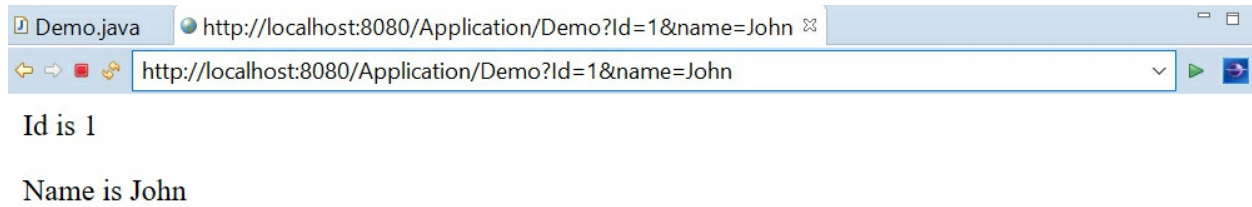
When we refresh the browser window in Eclipse, we will get the following output:



Id is null

Name is null

This is because we are not passing any Query String values, hence the default value of the ID and Name is null. If we append the Query String to the url, we get the following output:



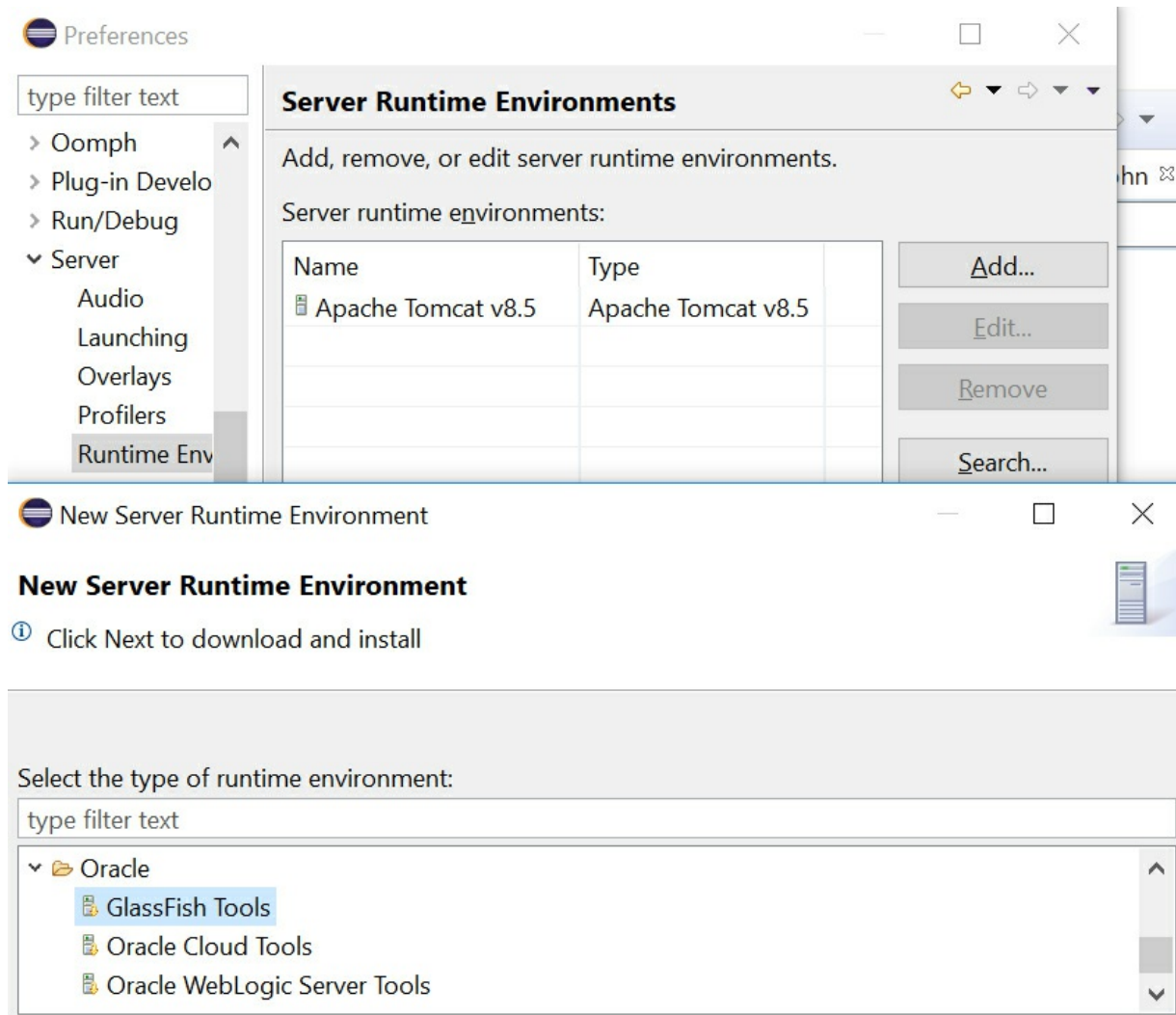
So from the above output we can see that once we enter the Query String parameters, we get the desired output.

11. Java Server Faces

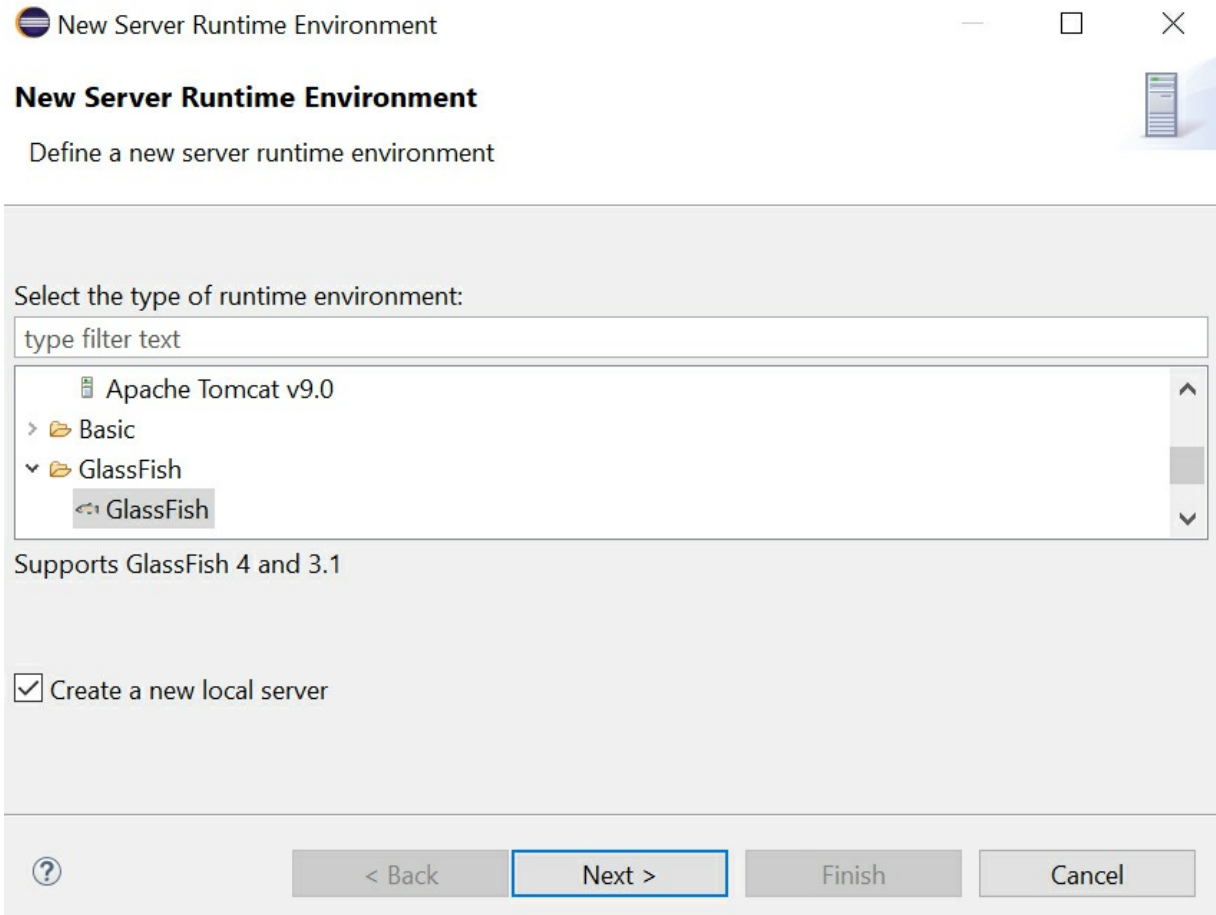
Java Server Faces (JSF) is used to develop component based user interfaces for web based applications. The Java Server Faces application runs on the server that processes the Java Server based application. The program itself can also contain JavaScript for client side interaction.

For the purpose of working with Java Server Faces, we need to use the GlassFish tools. GlassFish is a reference implementation of the Java Enterprise Edition and will help to build Java Server Faces programs. If we are using an IDE such as Eclipse, we need to add the GlassFish Tools to it. This can be done through the following steps:

- Go to 'Preferences'.
- Choose 'Server Runtime Environments'.
- Choose to add a new runtime environment. Here under Oracle, we can see the GlassFish Tools.



- If you don't have the GlassFish software, it can be downloaded from the following link: <https://javaee.github.io/glassfish/download>
- Once GlassFish has been installed, again go back to Preferences and Server Runtime Environment. Choose to add a runtime environment and ensure to choose 'Create a new local server'.



- Next we need to specify the location of the Java Runtime Environment and GlassFish folder.

New Server Runtime Environment

GlassFish

Define GlassFish runtime properties.

Name: GlassFish 4

GlassFish location: H:\Apps\glassfish\glassfish4\glassfish

Java location: C:\Program Files\Java\jdk1.8.0_101

? < Back Next > Finish Cancel

Once all this is done, we can start developing our first application.

11.1 Developing a Sample Application

Let's understand what goes into building a sample application. The following steps need to be in place:

Step 1

Create a web based project in the IDE.

Step 2

Create a Java class file that will act as a 'ManagedBean'. An example is shown below.

```
import javax.faces.bean.ManagedBean;
@ManagedBean
public class hello
{
```

```
    final String world = "Hello World!";  
    public String getworld()  
    {  
        return world;  
    }  
}
```

In the above program, we need to add an annotation for the class to ensure it is recognized as a ‘ManagedBean’. Then we define a property called ‘world’ and put the value as ‘Hello World’.

Step 3

In the ‘web.xml’ file, the following code should be present to ensure that this becomes a Java Face application.

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-  
app_3_1.xsd">  
    <context-param>  
        <param-name>javax.faces.PROJECT_STAGE</param-name>  
        <param-value>Development</param-value>  
    </context-param>  
    <servlet>  
        <servlet-name>Faces Servlet</servlet-name>  
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>  
        <load-on-startup>1</load-on-startup>  
    </servlet>  
    <servlet-mapping>  
        <servlet-name>Faces Servlet</servlet-name>  
        <url-pattern>/faces/*</url-pattern>  
    </servlet-mapping>  
    <session-config>  
        <session-timeout>  
            30  
        </session-timeout>  
    </session-config>  
    <welcome-file-list>  
        <welcome-file>faces/index.xhtml</welcome-file>  
    </welcome-file-list>  
</web-app>
```

Step 4

Create an xhtml file that will be used to render the application.

```
<html lang="en"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Facelets Hello World</title>
  </h:head>
  <h:body>
    #{hello.world}
  </h:body>
</html>
```

In the above program, we are referencing the static property from our class using the code line:

```
#{hello.world}
```

Once we run the application using the GlassFish Server, we will get the following output:

localhost:8080/Demo/

Hello World!

12. Java Database Connectivity (JDBC)

The Java Database Connectivity, or JDBC, API is used to interact with databases. With JDBC we can make connections to a database and work with the built-in SQL statements. First let's look at the setup. In our example, we are going to work with a MySQL database as our data source. The MySQL database setup can be downloaded from the link below:

<https://dev.mysql.com/downloads/installer/>

Once download and installed, let's create the following:

- A database called 'Person'.
- A table called 'Per' that has 2 columns. One column is titled 'ID' for the INT type, and the other is titled 'Name' for the varchar type.
- Add one item to the table as '1 John'.

Next we need to download the MySQL database connector for Java from the following location:

<https://www.mysql.com/products/connector/>

Once done, we need to add the jar file as an external jar for our project. Now let's add the following code to establish a connection to the database using JDBC and read the contents of the table.

Example 60: The following program shows how to read data from a database table.

```
import java.sql.*;
public class Sample
{
    public static void main(String args[]) throws Exception
    {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con=DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/Person","admin","password");
        Statement stmt=con.createStatement();

        ResultSet rs=stmt.executeQuery("select * from Per");

        while(rs.next())
```



```
        System.out.println(rs.getInt(1)+" "+rs.getString(2));
        con.close();
    }
}
```

The following things need to be noted about the above program:

- First we import the necessary SQL libraries.
- Then we use the 'DriverManager' class to get a connection to the database using the JDBC protocol. Here we mention the server name, port number, username and password to connect.
- Next we create a result set by executing a select query.
- Finally we use the 'recordset' to navigate through the records.

Once we run the program, we will get the following output:

1 John

We can also use JDBC to insert records into the database. Let's look at an example of this.

Example 61: The following program shows how to insert records into a table.

```
import java.sql.*;
public class Sample
{
    public static void main(String args[]) throws Exception
    {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con=DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/Person","admin","password");

        PreparedStatement stmt=con.prepareStatement("insert into per values(?,?)");
        stmt.setInt(1,2);
        stmt.setString(2,"Mark");

        int i=stmt.executeUpdate();
        System.out.println(" The number of records inserted is "+i);

        con.close();
    }
}
```

The following things need to be noted about the above program:

- We are using the 'PreparedStatement' here so that we can take dynamic values in our statement.
- We then use the 'set' methods to set the values of 'ID' and 'Name', which will be written to the database.
- Finally we execute the 'Update' statement to add the record to the database.

Once we run the program, we will get the following output:

The number of records inserted is 1

We can also get metadata about the table via JDBC. Let's look at an example of this.

Example 62: The following program shows how to get the table metadata.

```
import java.sql.*;
public class Sample
{
    public static void main(String args[]) throws Exception
    {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con=DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/Person","admin","password");

        PreparedStatement ps=con.prepareStatement("select * from per");
        ResultSet rs=ps.executeQuery();
        ResultSetMetaData rsmd=rs.getMetaData();

        System.out.println("Total columns: "+rsmd.getColumnCount());
        System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));
        System.out.println("Column Type Name of 1st column: "+rsmd.getColumnTypeName(1));

        con.close();
    }
}
```

Once we run the program, we will get the following output:

Total columns: 2

Column Name of 1st column: Id

Column Type Name of 1st column: INT

Another thing we can do is use JDBC to get the metadata about the database. Below is an example of this.

Example 63: The following program shows how to get the database metadata.

```
import java.sql.*;
import com.mysql.jdbc.DatabaseMetaData;

public class Sample
{
    public static void main(String args[]) throws Exception
    {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con=DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/Person","admin","password");

        DatabaseMetaData dbmd=(DatabaseMetaData) con.getMetaData();

        System.out.println("Driver Name: "+dbmd.getDriverName());
        System.out.println("Driver Version: "+dbmd.getDriverVersion());
        System.out.println("UserName: "+dbmd.getUserName());
        System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
        System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());

        con.close();
    }
}
```

Once we run the program, we will get the following output:

Driver Name: MySQL Connector Java

**Driver Version: mysql-connector-java-5.1.44 (Revision:
b3cda4f864902ffdde495b9df93937c3e20009be)**

UserName: admin@localhost

Database Product Name: MySQL

Database Product Version: 5.7.20-log

13. Android and Java

A great feature of Java is that it can be used to develop Android based applications. One reason why Java is favored as a programming language to develop Android apps, is because of its functionality as an object oriented programming language. To start developing Android based applications using Java, the first step is to download an IDE. Android Studio is a well renowned IDE for developing Android apps.

Some of the features of Android Studio are:

- Gradle-based build support.
- Android-specific refactoring and quick fixes.
- Lint tools to catch performance, usability, version compatibility and other problems.
- ProGuard and app-signing capabilities.
- Template-based wizards to create common Android designs and components.


The Android Studio and SDK are available at the following link:

<https://developer.android.com/studio/index.html>

Once you download and install Android Studio, you go ahead and build a new project. Make sure to choose the basic template to begin with. The steps below show how to create a project in Android Studio.

- First we start by giving our application a name.

 Create New Project



New Project

Android Studio

Configure your new project

Application name:

Company domain:

Package name:

☐ Include C++ support

- Select the form factors for the application. The API level will determine the compatibility of our app. The older we go, the more compatible our app will be, but we will lose some key features.

Select the form factors your app will run on

Different platforms may require separate SDKs

☒ Phone and Tablet

Minimum SDK API 16: Android 4.1 (Jelly Bean)

Lower API levels target more devices, but have fewer features available.

By targeting API 16 and later, your app will run on approximately **99.2%** of the devices that are active on the Google Play Store.

[Help me choose](#)

☐ Wear

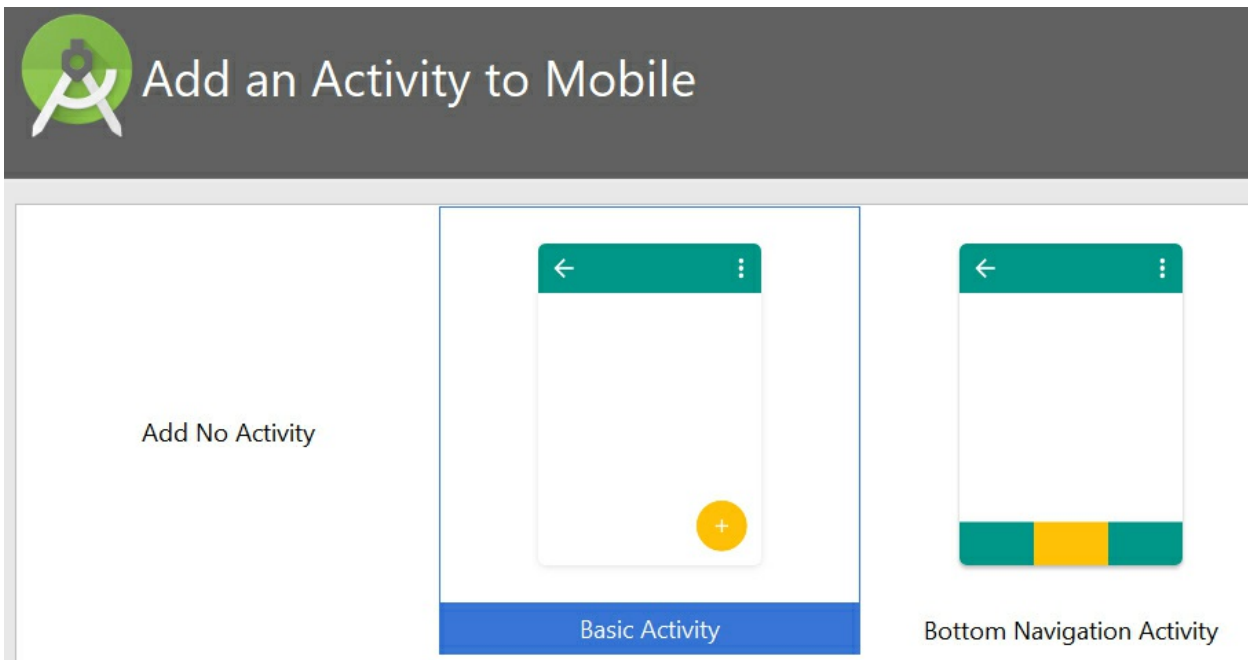
Minimum SDK API 21: Android 5.0 (Lollipop)

☐ TV

Minimum SDK API 21: Android 5.0 (Lollipop)

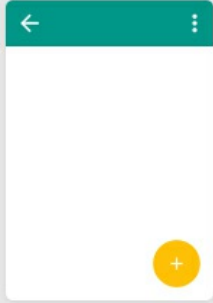
☐ Android Auto

- Choose the type of activity. We will start with a basic activity for our example.



- Name the different aspects of the activity.

Creates a new basic activity with an app bar.



Activity Name: MainActivity

Layout Name: activity_main

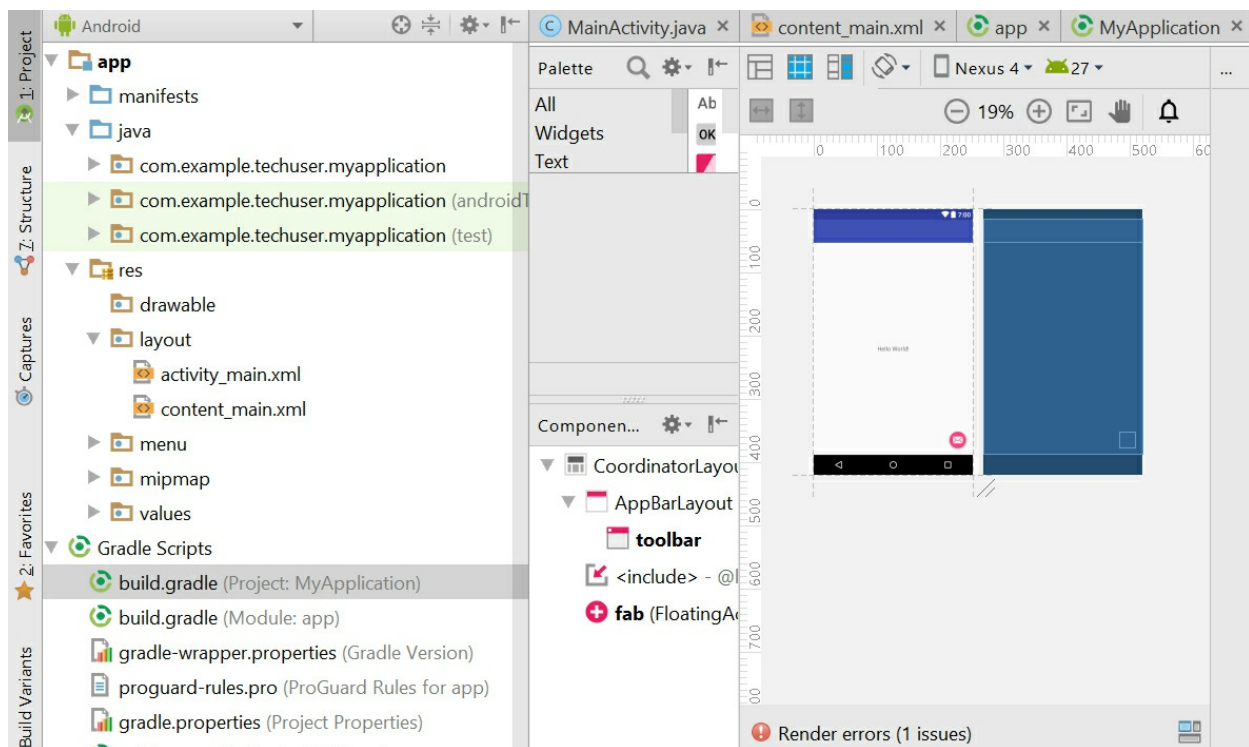
Title: MainActivity

Menu Resource Name: menu_main

☐ Use a Fragment

Basic Activity

- Once the project has been set up, we will get the interface below.



The MainActivity class will contain the following boiler template code:

```
package com.example.techuser.myapplication;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
```

```

import android.view.Menu;
import android.view.MenuItem;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
            }
        });
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}

```

Some of the key aspects of the above code are:

- The main class extends the ‘AppCompatActivity’ class that forms the basis for the activity bar for the mobile application.
- The ‘onCreateOptionsMenu’ method is used to create the menu for the mobile application.

To run the mobile application, we first need to create a virtual device as follows:

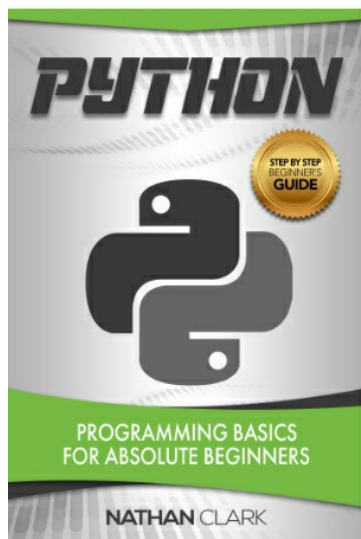
- Create a New Virtual Device from the menu.
- Choose the Device Type.
- Then we download a system image for the virtual device.

Once the image has been downloaded, we can go ahead and run a Java program. It will then run in the emulator.

Conclusion

We have unfortunately reached the end of this guide. But as I always say, it doesn't mean your Java journey should end here. Practice as much as possible. This book was written not only to be a teaching guide, but also a reference manual. So remember to always keep it near, as you venture through this wonderful world of programming.

If you enjoyed this guide, and this series, be sure to look into the other programming languages I cover, such as Python. It is a high-level language that is used in a variety of different applications. Python is best known for the simplicity of its programs. It requires less code to create a similar program in Python, compared to other languages. This makes it one of the most popular languages available today.



PYTHON

Programming Basics for Absolute Beginners

 **FREE Kindle Version with Paperback**

About the Author

Nathan Clark is an expert programmer with nearly 20 years of experience in the software industry.

With a master's degree from MIT, he has worked for some of the leading software companies in the United States and built up extensive knowledge of software design and development.

Nathan and his wife, Sarah, started their own development firm in 2009 to be able to take on more challenging and creative projects. Today they assist high-caliber clients from all over the world.

Nathan enjoys sharing his programming knowledge through his book series, developing innovative software solutions for their clients and watching classic sci-fi movies in his free time.