Java

The Quick Reference Manual

Last Edited: 10 April, 2018

Md. Saidul Hoque Anik
Lecturer, Dept of CSE, MIST

onix.hoque.mist @ gmail.com

Anika Binte Islam
Lecturer, Dept of CSE, MIST

anika.cse @ gmail.com

Zinia Sultana
Lecturer, Dept of CSE, MIST

sultana.hiramony @ gmail.com

Fatima Jannat
Lecturer, Dept of CSE, MIST

fj.jannat20 @ gmail.com

Table of Contents

Introduction

Think about someone - who never forgets, is extremely quick at making decisions, and never makes a mistake.

Done? Who is it?

What did you say? Your spouse? Maybe, but I was talking about the other one. A computer.

Computers are extremely fast at calculating, can remember exactly what you said two years ago, and will do exactly what you tell it to do without a mistake. It's a magnificent device. Our lives are full of problems. It would be really cool if we could use such a powerful device to solve our problems. And to communicate with it, we'll need to learn how to talk to the computers. So we'll need to know one or more programming languages.

We are assuming you are already familiar with C/C++ programming language. They are great for programming your PC. But computers are still big and heavy objects. And we often carry one or more almost equally powerful computers in our pockets, the smartphones. If we need to write programs for your PC and the smartphone at the same time, Java will be the top choice.

This manual works as a stepping stone for taking the first step into the world of Java. After going through this book, you'll have a rough idea on how to do the basic things like reading and writing in File/Console, making classes, using data structures in Java that you used to have in C/C++. For going further, you'll have to pick up an actual book written by a professional author.

So, let's begin! Bi'ismillah!

Chapter 0: Environment Setup
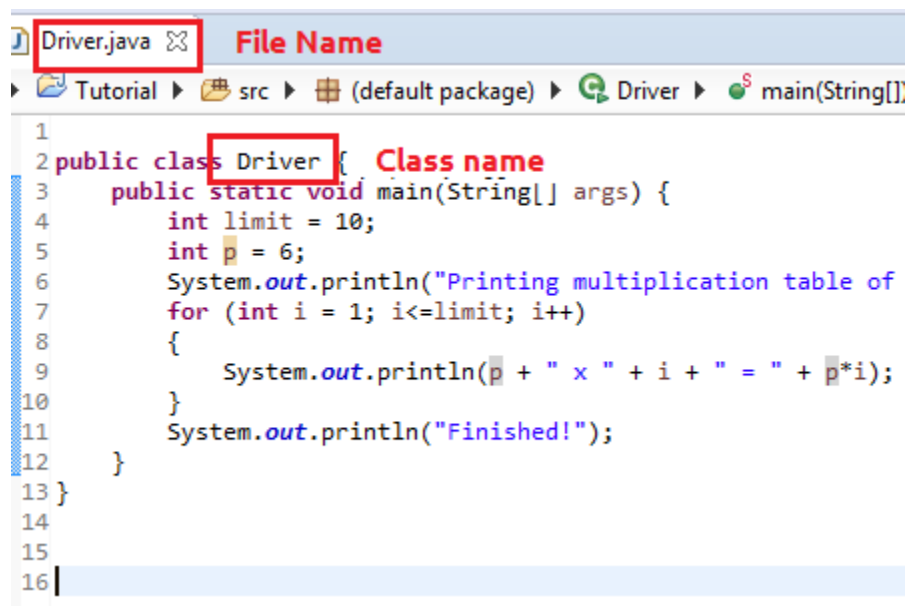
Before you start the manual, make sure-

1. You have Java installed in your computer. Go to run, type cmd. Then type:

   java -version

   If java version is not shown, your pc does not have java installed. Download JDK from the oracle's website.

2. Make sure you have an IDE installed. Netbeans or Eclipse are two popular choices. We shall be using Eclipse in this manual.
3. Open Netbeans/Eclipse. From File menu, choose New Java Project to create a java project. If your class file is not automatically created, create one by right clicking on the source package folder (**src** on Eclipse), and adding new java class.
4. Make sure your java class name and the file name is the same.
5. Make sure your java class has a main function with the exact signature as shown in the Figure below.

```
1
2 public class Driver {
3     public static void main(String[] args) {
4         int limit = 10;
5         int p = 6;
6         System.out.println("Printing multiplication table of
7         for (int i = 1; i<=limit; i++)
8         {
9             System.out.println(p + " x " + i + " = " + p*i);
10        }
11        System.out.println("Finished!");
12    }
13 }
14
15
16
```

Chapter 1: I/O and Control flow

Data types and I/O

Output

In order to print in console, Use `System.out.print()` or `System.out.println()`

```java
public class Driver {

    public static void main(String[] args) {
        int roll = 10;
        float gpa = 4.97f;   //f is to denote that the rational number is a float
        double pi = 3.1415926535897;

        System.out.println("Roll number is " + roll);
        System.out.println("GPA is " + gpa);
        System.out.println("Value of pi is " + pi);
    }
}
```

Input

To take input, we'll create an object of a scanner class. We'll tell the constructor of the scanner class to take input from the console by initializing it with the `System.in` object.

```java
import java.util.Scanner;

public class Driver {

    public static void main(String[] args) {
        int roll;
        float gpa;
        double pi;

        Scanner sc = new Scanner(System.in);

        roll = sc.nextInt();
        gpa = sc.nextFloat();
        pi = sc.nextDouble();

        System.out.println("Roll number is " + roll);
        System.out.println("GPA is " + gpa);
        System.out.println("Value of pi is " + pi);
    }
}
```

Note that, in order to use the scanner class, we'll have to add an extra import statement at the top. Your IDE will automatically/semi-automatically take care of it.

Control Flow

The control flow logics are similar to that of C/C++.

If-else

```java
import java.util.Scanner;

public class Driver {

    public static void main(String[] args) {
        int roll;
        Scanner sc = new Scanner(System.in);

        roll = sc.nextInt();
        if (roll % 2 == 1)
            System.out.println("Your Roll number is odd");
        else
            System.out.println("Your Roll number is even");

    }
}
```

If-else if-else

```java
import java.util.Scanner;

public class Driver {

    public static void main(String[] args) {
        int roll;
        Scanner sc = new Scanner(System.in);

        roll = sc.nextInt();
        if (roll <= 0)
            System.out.println("Roll number cannot be zero or negative");
        else if (roll % 2 == 1)
            System.out.println("Your Roll number is odd");
        else
            System.out.println("Your Roll number is even");

    }
}
```

For Loop

We use for loop for iterating for a definite amount of time.

```java
public class Driver {
    public static void main(String[] args) {
        int n = 20;
        for (int i = 0; i<n; i++)
        {
            System.out.println("Counting " + i);
        }
        System.out.println("Finished!");
    }
}
```

Here's a more useful example.

```java
public class Driver {
    public static void main(String[] args) {
        int limit = 10;
        int p = 6;
        System.out.println("Printing multiplication table of " + p);
        for (int i = 1; i<=limit; i++)
        {
            System.out.println(p + " x " + i + " = " + p*i);
        }
        System.out.println("Finished!");
    }
}
```

While loop

We use while loop when we do not know the exact number of iteration that needs to be performed.

```java
public class Driver {
    public static void main(String[] args) {
        int count = 0;
        int n = 1024;
        while (n>1)
        {
            n = n /2;
            count++;
        }
        System.out.println("1024 can be divided by two " + count + " times.");
    }
}
```

Chapter 2: Array

1D Array

In java, array can be created in either of the two following ways.

```java
public class Driver {
    public static void main(String[] args) {
        int [] array1 = new int[10];
        int [] array2 = {1, 2, 3, 4, 5};
    }
}
```

The first one allocates 10 integers in `array1`. The other one initializes five int elements accordingly in `array2`.

You can also take input and store into array in the following way.

```java
import java.util.Scanner;

public class Driver {
    public static void main(String[] args) {
        int [] array1 = new int[10];
        int [] array2 = {1, 2, 3, 4, 5};

        Scanner sc = new Scanner(System.in);

        for (int i = 0; i<array1.length; i++)
            array1[i] = sc.nextInt();


    }
}
```

Printing Array, for-each Loop
Accessing the elements of the array is similar to C/C++. Notice that the array variables have their own length property that gives the size of the array.

```java
public class Driver {
    public static void main(String[] args) {
        int [] array1 = new int[10];
        int [] array2 = {1, 2, 3, 4, 5};

        for (int i = 0; i<array2.length; i++)
            System.out.println("The " + (i+1) + "th element of array2 is " + array2[i]);
    }
}
```

However, the following example demonstrates an easier way to access the elements if you don't need the iterator or index.

```java
public class Driver {
    public static void main(String[] args) {
        int [] array1 = new int[10];
        int [] array2 = {1, 2, 3, 4, 5};

        for (int val : array2)
            System.out.println("array2 has " + val);
    }
}
```

Notice how the for loop is written. The syntax is the following.

```
for (<array element type> <holder variable> : <array name>)

    {     // holder variable will have each of the values of

    // the array one by one. }
```

2D Array

2D array can be created in a similar fashion.

```java
public class Driver {
    public static void main(String[] args) {
        int [][] array1 = new int[10][5];          // int[row][col]
        int [][] array2 = {{1, 2, 3}, {4, 5, 6}};  // row 1 -> {1, 2, 3}
                                                   // row 2 -> {4, 5, 6}

    }
}
```

Notice that 2D array creation requires two square brackets.

Populating 2D Arrays

Printing 2D array is similar to that in C/C++.

```java
for (int i = 0; i< 10; i++)
{
    for (int j = 0; j < 5; j++)
    {
        array1[i][j] = sc.nextInt();
    }
}
```

```
                    array[0][0]
                         ↑
array[0]  →  { 1, 2, 3 }
array[1]  →  { 4, 5, 6 }
                         ↓
                    array[1][2]
```

Notice that the first level index of an array is another 1D array (See Figure on top), and array itself is the array of these arrays. Thus, writing `array.length` gives us the height/row count of the matrix, and `array[0].length` gives us the width/column count of the matrix. So we can also access the elements in the following manner.

```
for (int i = 0; i< array1.length; i++)   //accessing the rows
{
    for (int j = 0; j < array1[0].length; j++)   //for i'th row, accessing each column
    {
        array1[i][j] = sc.nextInt();
    }
}
```

Printing 2D array

Printing the array can be done based on index.

```
public class Driver {
    public static void main(String[] args) {
        int [][] array1 = new int[10][5];          // int[row][col]
        int [][] array2 = {{1, 2, 3}, {4, 5, 6}};   // row 1 -> {1, 2, 3}
                                                    // row 2 -> {4, 5, 6}
        for (int i = 0; i< array2.length; i++)   //accessing the rows
        {
            for (int j = 0; j < array2[0].length; j++)   //for i'th row, accessing each column
            {
                System.out.print(array2[i][j] + " ");
            }
            System.out.print("\n");
        }
    }
}
```

However, we can also use for-each loop. The outer loop will hold the row arrays each time, and for each row, the inner loop will fetch the values of each column of that row individually.

```
for (int[] row : array2)     //accessing the rows
{
    for (int val : row) //for i'th row, accessing each column
    {
        System.out.print(val + " ");
    }
    System.out.print("\n");
}
```

Neat! Isn't it?

Chapter 3: Strings

Unlike C/C++, Strings are in-built data types in Java. However, they are not primitive data types, but object like arrays, and have their own property and methods such as length, charAt() etc.

String IO and comparison

In order to take a string as input, we use the nextLine() method of a scanner.

```java
import java.util.Scanner;

public class Driver {
    public static void main(String[] args) {
        String name;
        Scanner sc = new Scanner(System.in);
        System.out.println("Please enter your name: ");
        name = sc.nextLine();
        System.out.println("Hello " + name + ", nice to meet you!");
    }
}
```

Checking if two strings are equal is slightly different in Java. The following code will not work correctly.

```java
import java.util.Scanner;

public class Driver {
    public static void main(String[] args) {
        //This code will not work correctly
        String input;
        String password = "secret";
        Scanner sc = new Scanner(System.in);
        System.out.println("Please enter the password: ");
        input = sc.nextLine();
        if (input == password)
            System.out.println("Correct password!");
        else
            System.out.println("The password you entered is wrong!");
    }
}
```

Note that Strings are objects, so input and password are two references to the actual string objects. Therefore, checking (input == password) checks if these two point at the same location or not. This is not what we were planning to do. Instead, we want to compare the values/contents of the two strings. For this purpose, we'll need to use the equals() function.

```java
        System.out.println("Please enter the password: ");
        input = sc.nextLine();
        if (input.equals(password))
            System.out.println("Correct password!");
        else
            System.out.println("The password you entered is wrong!");
```

Common String Functions

Strings are objects, but immutable objects. What that means, is that, once they are created, they cannot be modified. When an operation is done on that string that tries to modify it, a new string is generated with the modification. The original string remains intact.

```java
public class Driver {
    public static void main(String[] args) {
        String sentence = "The quick brown fox";
        System.out.println(sentence.length());            //19
        System.out.println(sentence.substring(4, 9));     //quick
        System.out.println(sentence.startsWith("The"));   //true
        System.out.println(sentence.endsWith("fox"));     //true
        System.out.println(sentence.toUpperCase());       //THE QUICK BROWN FOX
        System.out.println("The position of 'fox' is in index " + sentence.indexOf("fox")); //16
        System.out.println(sentence.contains("red"));     //false
        sentence = sentence.replace("brown", "red");
        System.out.println(sentence.contains("red"));     //true
    }
}
```

Notice the line before the last one. `Sentence.replace()` takes the contents of the string, modifies it, and returns a new string with the modification. We are using the same variable `sentence` to hold the reference to the modified string. So in the last line, the modified string is being pointed by the variable '`sentence`', and the reference to the original string is lost.

Array of Strings

Array of strings is not so different from other arrays such as array of integers. As always, following two are the most common ways of declaring and initializing array of strings.

```java
String [] array_of_strings = new String[10];
String [] weekdays = {"Sunday", "Monday", "Tuesday", "Wednessday", "Thursday", "Friday", "Saturday"};
```

Printing the elements of array is easier with for-each loop.

```java
String [] weekdays = {"Sunday", "Monday", "Tuesday", "Wednessday", "Thursday", "Friday", "Saturday"};
for (String day : weekdays)
    System.out.println(day + " is a weekday");
```

the split() function

We can split a single string into array of strings based on the delimiter (the separator between words). For a sentence, the delimiter can be a single space, as the words are separated using spaces.

```java
String sentence = "The quick brown fox";
String [] word_list = sentence.split(" ");  //the words are now stored in word_list
for (String word : word_list)
    System.out.println(word + " is a word");
```

Following is another example where the delimiter is comma (,)

```java
String sentence = "Sunday,Monday,Tuesday";
String [] day_list = sentence.split(",");
for (String day : day_list)
    System.out.println(day + " is a weekday");
```

Type-Casting from String

Typecasting from string is a bit different from regular type-casting, because string is not a primitive data type.

```java
String s1 = "25";
String s2 = "2.7";

System.out.println(s1 + "5");    //255
System.out.println(s2 + "1.2"); //2.71.2

int a = Integer.valueOf(s1);
double b = Double.valueOf(s2);
System.out.println(a + 5);       //30
System.out.println(b + 0.4);     //3.1
```

Chapter 4: File I/O

Creating a File Object

In order to write in a file, first you need to create a file object. In the constructor, you'll need to pass the location of the file. If the file is in the same location as your project, you can only write down the file name (the relative path).

```java
import java.io.File;

public class Driver {
    public static void main(String[] args) {
        String s1 = "This is a line";
        String s2 = "This is another line";
        File f = new File("Sample.txt");
    }
}
```

Note that, the file need not exist. If a file is not found, it will automatically be created during writing. In case of reading, if the file is not found, FileNotFound exception will be thrown.

Writing in File

In order to write in a file object, A FileWriter object is required. In its constructor, the file object is passed down.

```java
import java.io.File;
import java.io.FileWriter;

public class Driver {
    public static void main(String[] args) {
        String s1 = "This is a line";
        String s2 = "This is another line";
        File f = new File("Sample.txt");

        FileWriter fr = new FileWriter(f);
        fr.write(s1 + "\n");
        fr.write(s2 + "\n");
    }
}
```

As you can see, we have created a FileWriter object named fr, and used the write() method to write two lines in it. However, the IDE is showing error as we haven't written the code inside any try-catch block. A lot of things could go wrong during File IO, so Java requires that you surround your FileWriter code with a try-catch block that will handle any IOException. Try-catch is not part of this manual, so using the autocomplete, we get the following code.

```
String s1 = "This is a line";
String s2 = "This is another line";
File f = new File("Sample.txt");

FileWriter fr;
try {
    fr = new FileWriter(f);
    fr.write(s1 + "\n");
    fr.write(s2 + "\n");
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

If we now go to our project folder, we'll see that a file has been created named Sample.txt. However, we'll be surprised to see that, the file is empty.

The flush() function

It turns out that the FileWriter object is a bit lazy. Although we have instructed it to write several lines, it has been waiting for more instructions with the plan in mind that it will collect more lines, and write the lines all together. However, that is not necessarily a bad thing. It is doing this only to increase efficiency.

So how do we let it know that we want it to write the lines immediately?

By calling the flush() method.

```
FileWriter fr;
try {
    fr = new FileWriter(f);
    fr.write(s1 + "\n");
    fr.write(s2 + "\n");
    fr.flush();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Now we'll see that our lines have been written properly.

The close() function

It is always a good idea to close the FileWriter object after the writings are done. It should be done so that the system knows that the file is no longer used by us, so it can allow other programs to have access to it.

```
FileWriter fr;
try {
    fr = new FileWriter(f);
    fr.write(s1 + "\n");
    fr.write(s2 + "\n");
    fr.flush();
    fr.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Reading line-by-line from file

Reading from file is very easy. For writing, we had used FileWriter Class. For reading, we shall use our very own Scanner class. But instead of initializing it with `System.in`, this time we shall initialize it with the file object.

```java
public class Driver {
    public static void main(String[] args) {
        File f = new File("Sample.txt");
        Scanner sc = new Scanner(f);
    }
}
```

Again, we need to surround it with try-catch blocks. Fortunately, our IDE can take care of that for us.

```java
File f = new File("Sample.txt");
try {
    Scanner sc = new Scanner(f);
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Now we want to read every line until the end of file is reached. Our scanner object can check that using `hasNextLine()` method. It returns true until no new readable next line is found. Then we can read each line using the `nextLine()` function.

```java
public class Driver {
    public static void main(String[] args) {
        File f = new File("Sample.txt");
        try {
            Scanner sc = new Scanner(f);
            while (sc.hasNextLine())
            {
                String line = sc.nextLine();
                System.out.println(line);
            }
            sc.close();
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Again, it's a good practice to close the scanner after reading from a file.

Chapter 5: Custom Classes

You might have noticed that we have been writing all of our codes in one method, the main method. The main method was inside the Driver class all along. Now it's time to create our very own custom class. But first, let's understand why we need to create another class.

Object Oriented Paradigm

Remember back in the days when you used to write codes in C? Suppose you want to keep track of the academic progress of your younger brother, who is now in primary school. You have decided to write a program to do that. In order to store his marks in Bengali, English and Math, you have declared three variables in your code. And when the total mark is needed, you just add them up, and show them using `printf`. So far so good.

What if we want to store the marks of 10 students?

We could use array of bengali, english and math. But that would scatter the data. It makes more sense if the number of the three subjects of a particular student is stored in a single place instead of storing all the bengali numbers together, all the english number together and so on.

So the smarter option is to use a structure.

A structure is used to hold multiple variables together in a single box. It's just creating your own compound variable that contains other primitive variables.

In java, instead of **struct**, we use something that is called the **class**. What makes the class powerful is that along with the variables, it can also hold functions that can run operation in the variables stored in it. For example, we can write a Student class that will hold the marks of Bengali, English and Math. Then we can write a getTotal() function that will return the summation of the marks stored inside.

To summarize, a class holds the blueprint of our custom data type, which can both hold information (in variable) and operation (as function). When a variable of that class is created, it is called an **object**.

The Student Class

So let's make our very own student class for starter. In order to make a class, right click on the source package folder, then click New, then select Java Class. The name of the class will be the name of the custom variable that you are creating, 'Student' in this case. As always, the name of the file must be same as the name of the class, and the access specifier must be public. If everything works out, you'll have something like this.

```java
public class Student {

}
```

What do we want each student to have? Holder variable for marks in Bengali, English and Math. What should be the data types? You guessed it right, they should be integers. So let's write them down.

```java
public class Student {
    int bengali;
    int math;
    int english;
}
```

Note that the name of the Class starts with capital letter (Student) and the name of the variables (bengali, math, english) starts with small letter. This is the convention.

Our student class is ready, now let's make some objects.

Declaring an object

How do we declare an integer? We write `int  a;` where `int` is the data type, and `a` is the name of the variable.

We declare an object in a similar fashion. In this case, our data type is called 'Student'. In order to make a Student object called s1, we shall write the following line in the Driver class.

```java
public class Driver {
    public static void main(String[] args) {
        Student s1 = new Student();
    }
}
```

Does it look familiar? It is almost similar to how we have created Strings, or Arrays.

Now that we have our student object s1, we can store the marks inside its own variables.

```java
public class Driver {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.bengali = 50;
        s1.english = 70;
        s1.math = 80;
    }
}
```

Member functions

Merely storing information may not be that useful, we want to perform operation on the information. So let's write a function inside our Student class that will calculate the total marks of that student.

```java
public class Student {
    int bengali;
    int math;
    int english;

    int getTotal()
    {
        int sum = bengali + math + english;
        return sum;
    }
}
```

getTotal() will add the three marks, and give us the summation. The summation is an integer value, that's why the return type of the function is `int`.

```java
public class Driver {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.bengali = 50;
        s1.english = 70;
        s1.math = 80;

        System.out.println("Total marks are: " + s1.getTotal());    //200

    }
}
```

As we can see, s1 contains the information as well as the functionalities together inside a single scope. This is known as object oriented programming. Objects are special types of variables that can hold other variables as well as functions. The instances of String class had variables such as length, and functions such as equals(), that's why they were called String objects. Similar is true for array objects.

Now create another method in the Student class called getAverage() that will return the average of the three marks. What will be the return type?

Constructor

If you look at the previous example, you'll see that each member variable of s1 had to be initialized individually. Writing such lines is often cumbersome. So we shall use a Constructor inside a class.

A Constructor is a function that is called when an object is created. It is used to initialize the variables that are inside the object. **This function has the same name as the name of the class.**

```java
public class Student {
    int bengali;
    int math;
    int english;

    Student(int b, int m, int e)
    {
        bengali = b;
        math = m;
        english = e;
    }

    int getTotal()
    {
        int sum = bengali + math + english;
        return sum;
    }
}
```

Now we can create objects more easily.

```java
public static void main(String[] args) {
    Student s1 = new Student(50, 70, 80);
    System.out.println("Total marks of s1 is: " + s1.getTotal());
}
```

The toString() function

Here's another thing about class. If we want to print the contents of an object in console in our own format, we'll need to create a toString() function in our class. **The function must be public,** and must return the string that needs to be printed.

```java
public class Student {
    int bengali;
    int math;
    int english;

    Student(int b, int m, int e)
    {
        bengali = b;
        math = m;
        english = e;
    }

    int getTotal()
    {
        int sum = bengali + math + english;
        return sum;
    }

    public String toString()
    {

    }
}
```

So what can we print? We can print the marks serially.

```java
public String toString()
{
    String output = "Marks of Bengali: " + bengali + ", english: " + english + ", math: " + math;
    return output;
}
```

Now checking the contents of a Student object is very easy.

```java
public class Driver {
    public static void main(String[] args) {
        Student s1 = new Student(70, 80, 90);
        System.out.println(s1); //s1.toString() function is automatically called
    }
}
```

Array of Objects

Creating an array of objects is easy. It similar to creating any other array.

```java
public class Driver {
    public static void main(String[] args) {
        Student[] student_list = new Student[10];

    }
}
```

Be careful. The elements of the student_list are not created yet, (only the array and the memory location is created), and they are assigned as null. So they are not ready to use, and if you try to access one of the element, you'll get a NullPointerException.

```java
public class Driver {
    public static void main(String[] args) {
        Student[] student_list = new Student[10];
        System.out.println(student_list[0].getTotal());
    }
}
```

```
<terminated> Driver (12) [Java Application] C:\Program Files\Java\jdk1.8.0_91\
Exception in thread "main" java.lang.NullPointerException
        at Driver.main(Driver.java:5)
```

We'll have to initialize each of the elements individually. You can always use a for loop.

```java
public class Driver {
    public static void main(String[] args) {
        Student[] student_list = new Student[10];
        student_list[0] = new Student(70, 80, 90);
        System.out.println(student_list[0].getTotal()); //240
    }
}
```

Chapter 6: Data Structures

As we have learned the bare minimum of Java, we are now ready to take on one of the most powerful aspects of this language, the data structures. Take a look at your moneybag or vanity bag. It contains a lot of pockets so that your money and other small scrap papers can be kept organized. Similarly, while writing programs, we need to handle a lot of information. In order to keep them organized, we need data structures. So far, we have only seen one data structure, the array. It's high time we explored a bit more.

Types of Data Structures

There are three types of data structures.

1. **Sequential**: Elements are sequentially stored, one after another after another. Each element can be identified using index. Array is a sequential data structure.
2. **Associative**: These data structures are used for checking if a value exists in the data structure or not. Suppose during a graph traversal, we want to quickly check whether we have visited a node or not. In this case, we can keep a set or bag. Set is an associative data structure. When we visit a node, we push the node into the set. Next time when we visit another node, we check if the node exists in our set/bag or not. Notice that, keeping a list of visited nodes is not a good idea here, as then we would have to go through all the elements of list from first to last to check if a node exists or not. Set is implemented using tree, so for existence checking, it is not necessary to traverse all the nodes.
3. **Special**: There are specially designed data structure for special purposes. Stack, Queue and Priority Queue are special data structures with distinct methods. They are internally designed using sequential or associative containers.

ArrayList

ArrayList is a sequential data structure. We have used array throughout the manual. However, once an array is created, its size cannot be modified. If we declare a large sized array, memory may be wasted if all of it is not used. On the other hand, declaring small sized array means we may run out of memory during runtime. So if we need a dynamic array, which will grow when we add new element, then ArrayList is just the thing for us.

Declaring an ArrayList

Declaring an arraylist is slightly different from a regular array. We must remember two things.

1. We'll have to mention the data type in an angle bracket.
2. We can no longer use `int`. Instead, we'll be using `Integer`. (Practically they are the same thing. The first one is a primitive data type, and the later one is a wrapper class, thus allows reference passing.)

So the process boils down to this.

```java
import java.util.ArrayList;

public class Driver {
    public static void main(String[] args) {
        ArrayList<Integer> lst = new ArrayList<>();
    }
}
```

Notice how the syntax is after the new keyword. We need not mention the data type a second time in the angle bracket.

List modifications – add, getSize, remove, search

Adding and accessing elements are very easy. Just call add() function to add. Print all elements using the handy for-each loop. **The size is found via size() function.**

```java
import java.util.ArrayList;

public class Driver {
    public static void main(String[] args) {
        ArrayList<Integer> lst = new ArrayList<>();
        lst.add(10);
        lst.add(20);
        lst.add(30);
        for (Integer i : lst)
            System.out.println(i);
    }
}
```

Removal is also possible. Just call remove() function and pass the index that you want removed.

```java
ArrayList<Integer> lst = new ArrayList<>();
lst.add(10);
lst.add(20);
lst.add(30);
lst.remove(1);
for (Integer i : lst)
    System.out.println(i);
```

Searching is done using indexOf() function. It takes the value to be searched as parameter.

```java
ArrayList<Integer> lst = new ArrayList<>();
lst.add(10);
lst.add(20);
lst.add(30);
int pos = lst.indexOf(30);
System.out.println("Index of 30 is " + pos);
```

Access by index

In order to access by index, you can no longer use the familiar [index] notation. You'll need to use get(index) method.

```java
ArrayList<Integer> lst = new ArrayList<>();
lst.add(10);
lst.add(20);
lst.add(30);
System.out.println("Element at pos 2 is " + lst.get(2));
```

HashSet

Hashset is an associative container. It is an implementation of Set/bag data structure. It has two fundamental properties.

1. Duplicate is not kept. Pushing a value second time in a set will not increase the count of that element.
2. As this is an associative container, we cannot sequentially access the elements in the order of insertion.

Initialization

Initialization of a HashSet is similar to that of ArrayList.

```java
import java.util.HashSet;

public class Driver {
    public static void main(String[] args) {
        HashSet<Integer> set = new HashSet<>();


    }
}
```

Adding and existence checking

The main task of set is checking whether an element exists or not. We can check it using contains() function. The addition is similar to ArrayList, the add() function.

```java
import java.util.HashSet;

public class Driver {
    public static void main(String[] args) {
        HashSet<Integer> set = new HashSet<>();

        for (int i = 10; i<=50; i = i + 10)
            set.add(i);

        if (set.contains(20))
            System.out.println("The set contains 20");
        else
            System.out.println("The set does not contain 20");
    }
}
```

Removal of element

In order to remove an element, we use remove() function.

```java
HashSet<Integer> set = new HashSet<>();

for (int i = 10; i<=50; i = i + 10)
    set.add(i);
set.remove(20);
if (set.contains(20))
    System.out.println("The set contains 20");
else
    System.out.println("The set does not contain 20");
```

Printing all the elements of a Set

In order to print the elements of a set, we first need to convert it into an array.

```java
HashSet<Integer> set = new HashSet<>();

for (int i = 10; i<=50; i = i + 10)
    set.add(i);
set.remove(20);

for (Object i : set.toArray())
    System.out.println(i);
```

HashMap

Have you ever used a phonebook? You have a person's name, and you are looking for his/her phone number? HashMap is just that. It 'maps' a **key** to its **value**. Given a key, it will quickly look up, and give away its value. In array, we look up using index. In hashmap, we can look up using other data types such as string, and even custom classes too!

HashMap is an associative data structure.

Initialization

As hashmap has two things (**key** and **value**), it has to be given two data type - the first one is the data type of **key**, and the second one is the data type of **value**.

```java
import java.util.HashMap;

public class Driver {
    public static void main(String[] args) {
        HashMap<String, Integer> phonebook = new HashMap<>();
    }
}
```

Adding and getting values

```java
HashMap<String, Integer> phonebook = new HashMap<>();
phonebook.put("Police", 999);
phonebook.put("FireBrigade", 919);

System.out.println("The phone number of police is : " + phonebook.get("Police"));  //999
System.out.println("The phone number of police is : " + phonebook.get("police"));  //null
```

Adding a value is slightly different in hashmap. You'll need the put() function to do it. To retrieve a value, use get() function.

If you enter a key that does not exist, it will return null.

Remove and Replace key-values

You can remove an entry from a hashmap using remove() function. You can replace an existing value of a key using replace() function.

```java
HashMap<String, Integer> phonebook = new HashMap<>();
phonebook.put("Police", 999);
phonebook.put("FireBrigade", 919);

phonebook.remove("FireBrigade");
phonebook.replace("Police", 777);

System.out.println("The phone number of police is : " + phonebook.get("Police"));    //777
System.out.println("The phone number of FireBrigade is : " + phonebook.get("FireBrigade")); //null
```

Existence Checking

You can explicitly check whether a key or a value exists or not using containsKey() and containsValue() function.

```java
HashMap<String, Integer> phonebook = new HashMap<>();
phonebook.put("Police", 999);
phonebook.put("FireBrigade", 919);

phonebook.remove("FireBrigade");
phonebook.replace("Police", 777);

System.out.println(phonebook.containsKey("Police"));    //true
System.out.println(phonebook.containsValue(999));       //false
```

Set of Keys and Values

You can also get the list of keys and values individually.

```java
HashMap<String, Integer> phonebook = new HashMap<>();
phonebook.put("Police", 999);
phonebook.put("FireBrigade", 919);

System.out.println(phonebook.values());
System.out.println(phonebook.keySet());
```

Printing all the elements of a HashMap

Printing all the elements of a hashmap is pretty complicated. Here's how to do it.

```java
import java.util.HashMap;
import java.util.Map;

public class Driver {
    public static void main(String[] args) {
        HashMap<String, Integer> phonebook = new HashMap<>();
        phonebook.put("Police", 999);
        phonebook.put("FireBrigade", 919);
        for (Map.Entry<String, Integer> e : phonebook.entrySet())
        {
            System.out.println(e.getKey() + ": " + e.getValue());
        }
    }
}
```

As we can see, the loop variable is an internal data structure of Map itself, which is called Entry.

Chapter 7: Special Data Structures

Specially designed data structures are often useful when imitating real world behavior. We'll discuss Stack, Queue, and PriorityQueue in this chapter. Finally we'll finish the book by explaining how we can create data structures with our custom class and custom sorting logic.

Stack

Imagine you are washing dishes. You are cleaning one dish, and putting it aside. Washing another dish, putting at the top of the previous one. Washing another one, putting it at the top of the last one. At the end, what you get is **stack** of plates.

Now if someone wants to take a dish, which one will he/she be able to take? The dish that was kept first? Or the dish that was kept last? Of course he/she will be able to take only the dish that was put at last. So the stack is a type of data structure that is **Last-in-first-out**. The last one to come, will leave the data structure first.

The standard operations of stack are the following.

1. Push: Put an element at the top of the stack.
2. Pop: Pick the top element from the top of the stack (it is removed from stack)
3. Top: See the top element without removing it from stack

Initializing a stack
Initializing a stack in Java is similar to other data structures.

Stack Operations

```java
import java.util.Stack;

public class Driver {
    public static void main(String[] args) {
        Stack<Integer> s = new Stack<>();
    }
}
```

Element can be pushed into stack using push() function. It can be picked using pop() function. The top value can be seen using peek() function. The size is known via size() function.

```java
Stack<Integer> s = new Stack<>();
s.push(10);
s.push(20);
s.push(30);

System.out.println("Poping the top element " + s.pop());    //30
System.out.println("Top of stack is now " + s.peek());    //20
System.out.println("Size of stack is " + s.size());    //2
```

Queue

Ever had to stand in a line at the bank? That's a queue. Someone comes in, and joins at the end of the queue. In contrast to Stack, the first person standing in the line gets served first. So queue is a **First-come-first-serve** data structure. New element joins at the end of the queue, and the oldest element is removed first.

The standard operations of a queue are the following.

1. **Enqueue:** Add a new element at the end of the queue.
2. **Dequeue:** Remove the first element from the front of the queue.
3. **Front:** Take a look at the front element of the queue without removing it.

Initialization of a Queue

Initialization of a Queue is a bit different in Java. It is initialized using LinkedList, as Queue itself is an Abstract Class (Something that is beyond the scope of this manual).

```java
import java.util.LinkedList;
import java.util.Queue;

public class Driver {
    public static void main(String[] args) {
        Queue<Integer> q = new LinkedList<>();
    }
}
```

Operations in a Queue

Enqueue operation is done using add() method. Dequeue is done using poll() method. peek() method gives the value at the front. Size is determined using size(), and isEmpty() returns true if Queue is empty.

```java
Queue<Integer> q = new LinkedList<>();
q.add(10);
q.add(20);
q.add(30);

System.out.println("Removing the oldest object: " + q.poll());  //10
System.out.println("Now " + q.peek() + " is at the front.");    //20
System.out.println("Size of Queue is " + q.size() + ", Empty = " + q.isEmpty()); //2, false
```

PriorityQueue

Suppose a group of students are standing in a line. Suddenly one of their teachers arrived. As the instructor is senior, he/she will stand directly in front of the line. In other words, he'll be given more **priority**.

Here's another example. If you have a lot of tasks at hand, you may start with the one that takes the least amount of time to complete. In this case, a task will get more priority if the estimated time to finish it is minimum.

Both of these examples have a queue. But unlike regular queues, dequeue operation in priority queue is not as simple as first-in-first-out. It is more like, more-priority-first-out. In the first example, the most senior a person will be at the beginning of the queue. In the second one, the task with least time required, will be at the first. So we can say that the first queue will be dequeued in descending order, and the second queue will be dequeued in ascending order.

Declaring a PriorityQueue in Java

Remember declaring a Queue? PriorityQueue declaration is the same, just initialize the queue with PriorityQueue class instead of LinkedList.

```java
import java.util.PriorityQueue;
import java.util.Queue;

public class Driver {
    public static void main(String[] args) {
        Queue<Integer> q = new PriorityQueue<>();

    }
}
```

PriorityQueue Operations

The default sort-order of a priority queue in java is ascending order (So it's internally a Min-Heap). The operations are similar to that of a Queue.

```java
Queue<Integer> q = new PriorityQueue<>();
q.add(70);
q.add(30);
q.add(100);

System.out.println("Removing the smallest object: " + q.poll());      //30
System.out.println("Now " + q.peek() + " is at the front.");      //70
System.out.println("Size of Queue is " + q.size() + ", Empty = " + q.isEmpty()); //2, false
```

The insertion order is 70, 30, 100. But it will be accessed in ascending order. So access order will be 30, 70, 100. After polling, 30 is dequeued. Therefore, 70 is at the front.

Java Comparator Class

In the previous section, we have seen how PriorityQueue can sort the inputs in ascending order. But what if we want them in descending order? What if our elements in the queue is not integer, but strings, and we want them sorted by the length of the elements? In that case, we'll have to define our own comparator, a special type of class that will define how our priority queue will be sorted. First, let's create one that dequeues in descending order.

Creating our own comparator class

Let's add a new java class file in our project. You know the drill. Right click on the package icon, click 'new', and then 'class'. Let's name our class as DescendingComparator. You can use a shorter name if you want, but it's more benefical to have variable names that talk to you.

Now we want our class to be a comparator class. So we'll have to add an extra part in our class. Change the declaration of your class so that it looks like the following figure.

```java
import java.util.Comparator;

public class DescendingComparator implements Comparator<Integer> {

}
```

Notice that we have written Integer inside the angle bracket, because our priority queue elements are of Integer type.

Now IDE is showing an error, because we'll have to define our logic on how to sort the the queue. We'll have to write a special function inside this comparator class known as compare function. Use your IDE suggestion to auto-generate the compare function skeleton.
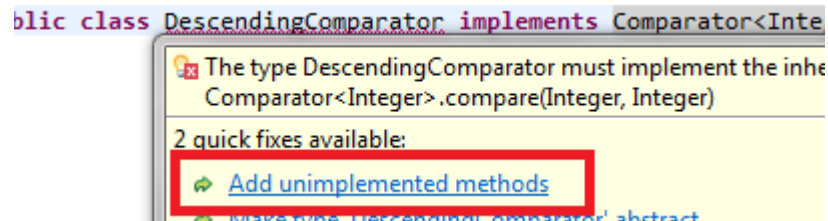
The compare() function

After auto-generating, your code should look like this. If you are not able to auto-generate, just write down the code from the figure below.

```java
import java.util.Comparator;

public class DescendingComparator implements Comparator<Integer> {

    @Override
    public int compare(Integer o1, Integer o2) {
        // TODO Auto-generated method stub
        return 0;
    }

}
```

Notice that the compare function has two parameters, o1 and o2. Both are Integer, because we have mentioned Integer inside the angle bracket after Comparator. Let's understand how we can use these parameters.

Have you ever written a bubble-sort code? In that algorithm, each element is compared with another one, and if the element i and element j are not in the right place, they are **swapped**.
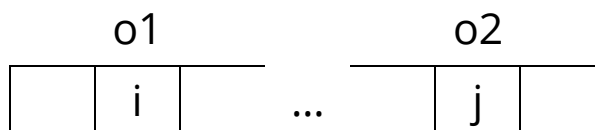
In the compare function, we shall return a negative value (maybe -1) if o1 and o2 are at the right place, and they don't need swapping.

What does it mean to be at the right place? O1 is the element at the front, and o2 is the element after o2. If we are sorting in descending order, the element in the former position should be larger than the element that comes after it. So o1 and o2 will be at the right place if o1 > o2, implying that they should not be swapped.

So if o1 > o2, we are saying that- 'negative, we don't need to swap in this case.' Therefore, we return a negative value.

On the other hand, if o1 and o2 are in the wrong place, we'll need to swap them. In case of descending order sort, o1 and o2 are in wrong place if the former element is smaller than the later one, i.e. o1 < o2. In that case, we shall say that, 'yes, it's positive, we need to swap', and return a positive value (+1).

If none of the cases occur, there could be only one case left, that o1 and o2 are equal. It's convention to return zero if both elements are equal. Let's write down the code.

```java
import java.util.Comparator;

public class DescendingComparator implements Comparator<Integer> {

    @Override
    public int compare(Integer o1, Integer o2) {
        //code for descending order sort
        if (o1 > o2)
        {
            return -1;   //o1 > o2, so no need to swap,
                         //so return negative, we don't need to sort
        }
        else if (o1 < o2)
        {
            return 1;    //o1 < o2, not in descending order,
                         //so return positive, 'yes! we're positive that we need to swap
        }
        else
            return 0;    //in case of o1 == o2, it's convention to return 0
    }

}
```

PriorityQueue in descending order

Now that our comparator is ready, let's integrate it into our PriorityQueue. Come back in Driver class. Modify the declaration so that the code look like the following figure.

```java
public static void main(String[] args) {
    DescendingComparator dc = new DescendingComparator();
    Queue<Integer> q = new PriorityQueue<>(10, dc);
    q.add(70):
```

In the first line at main function, we make an object of the comparator that we have just created. The object is passed as the second argument in PriorityQueue constructor. The first parameter of PriorityQueue takes the initial capacity of the queue, we can put any non-zero positive value in it.

Now if we run the code, we'll see a different output. Our PriorityQueue has become a Max-Heap.

```java
public static void main(String[] args) {
    DescendingComparator dc = new DescendingComparator();
    Queue<Integer> q = new PriorityQueue<>(10, dc);
    q.add(70);
    q.add(30);
    q.add(100);
    System.out.println("Removing the *largest* object: " + q.poll());    //100
    System.out.println("Now " + q.peek() + " is at the front.");    //70
    System.out.println("Size of Queue is " + q.size() + ", Empty = " + q.isEmpty()); //2, false
}
```

PriorityQueue with Custom Object

We'll conclude our manual by designing a PriorityQueue with our own class instead of a simple Integer class. Our Priority Queue will have Person objects sorted in ascending order. Each person will have his name (String), and his/her height. So let's design the class.

Designing the Person Class

Add a new class in your project named Person. Decorate the class in the following way.

```java
public class Person {
    String name;
    int height;
    Person (String n, int h)
    {
        name = n;
        height = h;
    }

    public String toString()
    {
        String ret = name + " (" + height + ")";
        return ret;
    }
}
```

We have added a constructor for easy initialization, and toString() method for easy printing.

Designing PersonComparator

Our PriorityQueue will have person objects inside. But we must tell it how it will compare a person with another. In order to sort the persons in ascending order, we'll have to make sure if the person in the front has less height than the person at the back. In order to do that, let's add a new java class, name it PersonComparator, and generate the compare function.

```java
import java.util.Comparator;

public class PersonComparator implements Comparator<Person>{

    @Override
    public int compare(Person o1, Person o2) {
        // TODO Auto-generated method stub
        return 0;
    }

}
```

Notice that, inside angle bracket, we have written 'Person', because we'll be comparing one Person with another. So, what should be the logic? When should we return minus one?

The compare function will return negative if the height of the former person is less than that of the later person. The height of the former person is given by o1.height (Because o1 is a Person object, so it will have its height property). Similarly, the height of the later person is given by o2.height. We shall return -1 if (o1.height < o2.height).

We know the rest. We shall return positive if two person are not in correct sequence. That means we shall return +1 if o1.height > o2.height. And we shall return 0 if both have equal heights. If you write down the code, it will look like the following.

```java
import java.util.Comparator;

public class PersonComparator implements Comparator<Person>{

    @Override
    public int compare(Person o1, Person o2) {
        if (o1.height < o2.height)
            return -1;
        else if (o1.height > o2.height)
            return 1;
        else
            return 0;
    }

}
```

Applying PersonComparator

Now let's declare a PriorityQueue in our main function that will hold Person objects, and sort them using our own PersonComparator.

```java
import java.util.PriorityQueue;
import java.util.Queue;

public class Driver {
    public static void main(String[] args) {
        PersonComparator pc = new PersonComparator();
        Queue<Person> q = new PriorityQueue<>(10, pc);

    }
}
```

Kindly notice that PersonComparator declaration does not have an angle bracket.

Let's add a few persons and see if the PriorityQueue works.

```java
PersonComparator pc = new PersonComparator();
Queue<Person> q = new PriorityQueue<>(10, pc);

q.add(new Person("Person A", 63));
q.add(new Person("Person B", 76));
q.add(new Person("Person C", 51));

while (q.isEmpty() == false)
    System.out.println(q.poll());      //Person C (51)
                                       //Person A (63)
                                       //Person B (76)
```

As you can see, the PriorityQueue is working as expected, extracting the persons in the ascending order of their heights.

Chapter 8: Exercise

Implement the following pseudocodes in Java to increase your coding skill. Write necessary codes to check the correctness of your implementation. Skip the ones that seem difficult. Consult the Java Manual as well as the Internet for code snippets. The first solution is given for you.

1. Bubble Sort

```
procedure bubbleSort( A : list of sortable items )
    n = length(A)
    repeat
        swapped = false
        for i = 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap( A[i-1], A[i] )
                swapped = true
            end if
        end for
    until not swapped
end procedure
```
Source: https://en.wikipedia.org/wiki/Bubble_sort#Pseudocode_implementation

Sample Solution
(Source: http://www.algorithmist.com/index.php/Bubble_sort)

```java
import java.util.Scanner;
public class BubbleSort
{      static void bubblesort(Integer[] ar)
       {
            for (int i = 0 ; i < ar.length ; i++)
                for (int j = 1 ; j < ar.length - i ; j++)
                {
                    if ( ar[j-1] > ar[j] )
                    {
                        int temp = ar[j-1];
                        ar[j-1] = ar[j];
                        ar[j] = temp;
                    }
                }
       }
       public static void main(String[] args){
            Scanner in = new Scanner(System.in);
            int n = in.nextInt();
            Integer[] ar = new Integer[n];
            for (int i = 0; i < n; i++)
                ar[i] = in.nextInt();
            bubblesort(ar);
            for (Integer v : ar)
```

```
            System.out.print(v+" ");
        System.out.println("");
    }}
```

2. Insertion Sort

```
for i from 1 to N
    key = a[i]
    j = i - 1
    while j >= 0 and a[j] > key
        a[j+1] = a[j]
        j = j - 1
    a[j+1] = key
```
Source: http://www.algorithmist.com/index.php/Insertion_sort

3. Merge Sort

```
func mergesort( var a as array )
     if ( n == 1 ) return a

     var l1 as array = a[0] ... a[n/2]
     var l2 as array = a[n/2+1] ... a[n]

     l1 = mergesort( l1 )
     l2 = mergesort( l2 )

     return merge( l1, l2 )
end func

func merge( var a as array, var b as array )
     var c as array

     while ( a and b have elements )
         if ( a[0] > b[0] )
             add b[0] to the end of c
             remove b[0] from b
         else
             add a[0] to the end of c
             remove a[0] from a
     while ( a has elements )
         add a[0] to the end of c
         remove a[0] from a
     while ( b has elements )
         add b[0] to the end of c
         remove b[0] from b
     return c
end func
```
Source: http://www.algorithmist.com/index.php/Merge_sort

4. Quick Sort

```
Quicksort(A as array, low as int, high as int){
    if (low < high){
        pivot_location = Partition(A,low,high)
        Quicksort(A,low, pivot_location)
        Quicksort(A, pivot_location + 1, high)
    }
}
Partition(A as array, low as int, high as int){
    pivot = A[low]
    leftwall = low

    for i = low + 1 to high{
        if (A[i] < pivot) then{
            swap(A[i], A[leftwall + 1])
            leftwall = leftwall + 1
        }
    }
    swap(pivot,A[leftwall])

    return (leftwall)}
```

Source: http://www.algorithmist.com/index.php/Quicksort

5. Stack

Implement a Stack class with the following methods:

  i)   Void Push(int val)
  ii)   Int Pop()
  iii)   Int top()
  iv)   Int size()
  v)   void Print()

6. Queue

Implement a Queue class with the following methods:

  i)   void enqueue(int val)
  ii)   int dequeue()
  iii)   int front()
  iv)   int size()
  v)   void print()

7. Single/Double Linked List

Implement the Single/Double Linked list with the following methods:

- i)   push_front(int val) // add at the front of the list
- ii)   push_back(int val)   //add at the tail of the list
- iii)   add middle (int index, int val)
- iv)   remove(int index)
- v)   remove(int val)
- vi)   int find(int val)
- vii)   int size()
- viii)   void print()

8. Graph Data Structure

Design a graph datastructure with the following methods:

- i)   addVertex(int v)
- ii)   addEdge(int v1, int v2)
- iii)   getAdjacentVertex(int v)

9. Depth-First Search

- i)   Implement the following pseudocode:

```
procedure DFS(G,v):
      label v as discovered
      for all edges from v to w in G.adjacentEdges(v) do
           if vertex w is not labeled as discovered then
                recursively call DFS(G,w)

procedure DFS-iterative(G,v):
      let S be a stack
      S.push(v)
      while S is not empty
          v = S.pop()
          if v is not labeled as discovered:
               label v as discovered
               for all edges from v to w in G.adjacentEdges(v) do
                    S.push(w)
```

Source: https://en.wikipedia.org/wiki/Depth-first_search#Pseudocode

- ii)   Test it with your sample input.
- iii)   Print the DFS tree (Similar to the tree below)
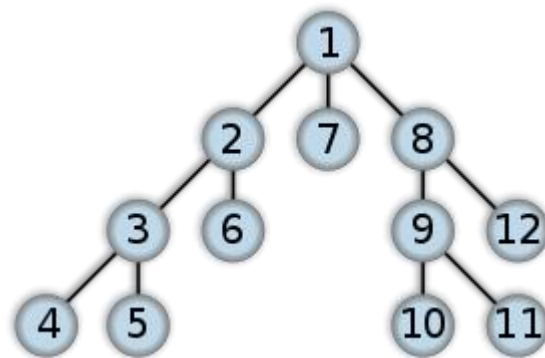
10. Breadth-first Search

i)   Implement the following pseudocode:

```
BFS (G, s)   //Where G is the graph and s is the source node
      let Q be queue.
      Q.enqueue(s) //Inserting s in queue until all its neighbour
                       vertices are marked.
      mark s as visited.
      while ( Q is not empty)
                 //Removing that vertex from queue,whose neighbour
                  will be visited now
          v  =  Q.dequeue()
             //processing all the neighbours of v
         for all neighbours w of v in Graph G
              if w is not visited
                     Q.enqueue(w) //Stores w in Q to further
                                    visit its neighbour
                 mark w as visited.
```

Source: https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

ii)   Test it with your sample input.

iii)   Print the BFS tree

So, where do I go from here?

If you have reached this chapter, you have surely come a long way. It was probably not easy, but surely it was fun. This manual covers some critical parts of Java that are regularly used in day-to-day programming. However, in order to become proficient in Java, you must explore the following topics.

1. How inheritance works
2. How to use access specifiers
3. Why should we use abstract class
4. How to design a GUI using Java swing
5. How to make inner classes
6. How networking works in java
7. Multithreaded programming

Any standard textbook/reference of Java should cover these topics. Explore them when you need them.