Atma Ram Sanatan Dharma College
University of Delhi

Programming in Java

Assignment for Paper Code 32341201

Submitted By
Sudipto Ghosh
College Roll No. 19/78003
BSc (Hons) Computer Science

Submitted To
Dr Parul Jain
Department of Computer Science

Assignment (Week 1)

Q1 What will be the output of the following programs?

(a)
```
class A {
    public A() {
        System.out.println("Class A constructor");
    }
}
class B extends A {
    public B() {
        System.out.println("Class B constructor");
    }
}
class C extends B {
    public C() {
        System.out.println("Class C constructor");
    }
}
public class MainClass {
    public static void main(String[] args) {
        C c = new C();
    }
}
```

(a) >java MainClass
```
Class A constructor
Class B constructor
Class C constructor
```

(b)
```
class A {
    String s = "class A";
}
class B extends A {
    String s = "class B";
    {
        System.out.println(super.s);
    }
}
class C extends B {
    String s = "class C";
    {
        System.out.println(super.s);
    }
}
```

```
public class MainClass {
  public static void main(String[] args) {
    C c = new C();
    System.out.println(c.s);
  }
}
```

(b) > java MainClass
Class A
Class B
Class C

(c)
```
class CommLine {
  public static void main(String[] args) {
    for (int i=0; i < args.length; i++)
      System.out.println("args["+i+"]: " + args[i]);
  }
}
```

(c) > java CommLine Alpha Beta Gamma
args[0]: Alpha
args[1]: Beta
args[2]: Gamma

Q2. Can abstract class have constructors in Java?

Ans. Yes, an abstract class can have constructors in Java. We can explicitly provide a constructor to abstract class. Otherwise, the compiler will append a default constructor accepting no arguments to the abstract class. This is true for all classes and it also applies to an abstract class.

Abstract classes by definition, can have one or more abstract methods. They cannot be instantiated but can be inherited by other classes. It is for this reason, that the constructor in the abstract class is invoked while calling the constructor of the subclass. The subclass can make calls to the constructor in the abstract class explicitly using the super() method.

Example:
```
abstract class Server {
  protected final String name;
  public Server(String name) {
    this.name = name;
  }
```

```
    public abstract boolean start();
}

class Tomcat extends Server {
    public Tomcat(String name) {
        super(name);
    }
    @Override
    public boolean start() {
        System.out.println(this.name+" started successfully");
        return true;
    }
}
public class MainClass {
    public static void main(String[] args) {
        Server server = new Tomcat("Apache Tomcat");
        server.start();
    }
}
```

OUTPUT

```
> java MainClass
Apache Tomcat started successfully.
```

Q3. Create an abstract class Parent with a method message(). It has two subclasses each having a method with the same name message() that prints "This is the first subclass" and "This is the second subclass" respectively. Call the methods message by creating an object for each subclass.

Ans.

```
abstract class Parent {
    abstract void message();
}
class FirstSubClass extends Parent {
    void message() {
        System.out.println("This is the first subclass");
    }
}
class SecondSubClass extends Parent {
    void message() {
        System.out.println("This is the second class");
    }
}
```

```
public class Main {
    public static void main (String [] args) {
        FirstSubClass f = new FirstSubClass ();
        SecondSubClass s = new SecondSubClass ();
        f.message ();
        s.message ();
    }
}
```

OUTPUT

```
> java Main
This is the first subclass
This is the second subclass
```

Q4. An abstract class has a constructor which prints "This is constructor of abstract class", an abstract method named a_method() and a non-abstract method which prints "This is a normal method of abstract class".
A class SubClass inherits the abstract class and has a method named a_method() which prints "This is abstract method".
Now create an object of SubClass and call the abstract method and the non-abstract method.
Analyse the result.

Ans.

```
abstract class AbstractClass {
    AbstractClass() {
        System.out.println("This is constructor of abstract class");
    }
    abstract void a_method();
    void normal_method() {
        System.out.println("This is a normal method of abstract class");
    }
}
class SubClass extends AbstractClass {
    void a_method() {
        System.out.println("This is abstract method");
    }
}
public class Main {
    public static void main (String[] args) {
        SubClass s = new SubClass();
        s.a_method();
        s.normal_method();
} }
```

OUTPUT

This is constructor of abstract class
This is abstract method
This is a normal method of abstract class

Q5. ~~Write Java code.~~

Analysis

When the SubClass is instantiated, the constructor of its parent/superclass is called. In this case, the Superclass is Abstract Class which prints the line "This is constructor of abstract class" to the console which forms the output of the code in the AbstractClass. constructor. This also proves that AbstractClass is inherited by SubClass.

SubClass provides a concrete definition of a_method() in Abstract class. The definition of a_method() causes the program to print "This is abstract method".

Definition of normal_method() in the Abstract Class prints the line "This is the normal method of abstract class".

Q5. Write Java code to find whether a number is prime or not where the number is accepted from the command line.

Ans.
```
import java.io.*;
public class Main {
    static int n;
    static boolean checkPrime(int n) {
        if (n<2)
            return false;
        for (int i=2; i*i<=n; i++)
            if (n%i==0)
                return false;
        return true;
    }
}
public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(
                                    System.in));
    System.out.print("Enter an integer: ");
    n= Integer.parseInt(br.readLine());
    System.out.println(n+" is "+(checkPrime(n)==true?"":"not")
                        +" a prime number.");
}
```

OUTPUT

```
>java Main2
Enter an integer: 2
2 is a prime number.
>java Main 2
Enter an integer: 4
4 is not a prime number.
```

Topic _____ Date _____

ASSIGNMENT. WEEK(2)

Q1 What will be the output of the following programs?

→ CODE

```
interface A {
    void myMethod();
}
class B {
    public void myMethod() {
        System.out.println("My Method");
    }
}
class C extends B implements A{

}
class MainClass {
    public static void main(String[] args) {
        Aa = new C();
        a.myMethod();
    }
}
```

● OUTPUT

```
My Method
```

Topic _____ Date _____

CODE

```
interface P{
    String p = "PPPP";
    String methodP();
}
interface Q extends P {
    String q = "QQQQ";
    String methodQ();
}
class R implements P, Q {
    public String methodP() {
        return q+p;
    }
    public String methodQ() {
        return p+q;
    }
}
public class MainClass{
    public static void main (String[] args) {
        R r = new R();
        System.out.println(r.methodP());
        System.out.println(r.methodQ());
    }
}
```

OUTPUT

```
QQQQPPPP
PPPPQQQQ
```

Topic _____ Date _____

Q2 Create a class TwoDim which contains private members as x and y coordinates in package P1. Define the default constructor, a parameterized constructor and override toString() method to display the coordinates. Now reuse this class and in package P2, create another class ThreeDim, adding a new dimension z as its private member. Define the constructors for the subclass and override toString() method in the subclass also, write appropriate methods to show dynamic method dispatch. The main() function should be in a package P.

➡ **CODE**

```
/**** P1/TwoDim.java ****/
package P1;
public class TwoDim {
  private int x;
  private int y;
  public TwoDim() {
    this.x = 0;
    this.y = 0;
  }
  public TwoDim(int x, int y) {
    this.x = x;
    this.y = y;
  }
  @Override
  public String toString() {
    return "Coordinate: x=" + x + " y=" + y;
  }
}
```

Topic_____ Date_____

```
/**** P2/ThreeDim.java ****/
package P2;
import P1.*;
public class ThreeDim extends TwoDim {
   private int z;
   public ThreeDim() {
      super(0,0);
      this.z = 0;
   }
   public ThreeDim(int x, int y, int z) {
      super(x,y);
      this.z = z;
   }
   @Override
   public String toString() {
      return super.toString() + " z = " + z;
   }
}

/**** P/Main.java ****/
package P;
import P1.*;
import P2.*;
public class Main {
   public static void main(String[] args) {
      TwoDim ref;
```

Topic _____ Date _____

```
        ref = new TwoDim (1,2);
        System.out.println(ref);
        ref = new ThreeDim (3,4,5);
        System.out.println (ref);
    }
}
```

- OUTPUT (on running java P.main)

```
Coordinate :  x = 1  y = 2
Coordinate :  x = 3  y = 4  z = 5
```

Topic _____ Date _____

Q3 Define an abstract class Shape in package P1. Inherit two more classes: Rectangle in package P2 and Circle in package P3. Write a program to ask the user for the type of shape and then using the concept of dynamic method dispatch, display the area of the appropriate subclass. Also write appropriate methods to read the data. The main() function should not be in any package.

→ CODE

```
/**** P1/Shape.java ****/
package P1;
public abstract class Shape {
  protected abstract void getData() throws java.io.IOException;
  public abstract double area() throws java.io.IOException;
}
```

```
/**** P2/Rectangle.java ****/
package P2;
import java.io.*;
import P1.*;
public class Rectangle extends Shape {
   private double length;
   private double breadth;
   protected void getData() throws IOExceptions {
      BufferedReader br = new BufferedReader(new InputStreamReader(
         System.in
      ));
```

Topic _____ Date _____

```
        System.out.print("Enter Length of Rectangle: ");
        length = Double.parseDouble(br.readLine());
        System.out.print("Enter Breadth of Rectangle: ");
        breadth = Double.parseDouble(br.readLine());
    }
    public double area() throws IOException {
        getData();
        return length * breadth;
    }
}
```

```
/**** P3/Circle.java ****/
package P3;
import java.io.*;
import P1.*;
public class Circle extends Shape {
    private double radius;
    protected void getData() throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(
            System.in
        ));
        System.out.print("Enter Radius of Circle: ");
        radius = Double.parseDouble(br.readLine());
    }
    public double area() throws IOException {
        getData();
        return Math.PI * radius * radius;
    }
}
```

Topic _____ Date _____

```
/**** Main.java ****/
import java.io.*;
import P1.*;
import P2.*;
import P3.*;
public class Main {
  static int getShapeType() throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(
      System.in
    ));
    System.out.println("====\n SHAPE TYPE\n====");
    System.out.println("(1) Rectangle\n(2) Circle");
    System.out.print("Enter Choice: ");
    return Integer.parseInt(br.readLine());
  }
  public static void main(String[] args)throws IOException {
    Shape ref;
    boolean flag = false;
    while(!flag) {
      switch(getShapeType()) {
        case 1:
          flag = true;
          ref = new Rectangle();
          System.out.println("Area: " + ref.area()
                             + " sq units");
          break;
```

Topic _____ Date _____

```
                case 2:
                        flag = true;
                        ref = new Circle();
                        System.out.println("Area: " + ref.area() +
                                                "sq units");
                        break;
                default:
                        System.err.println("Invalid Option");
                        Break;
            }
        }
    }
}
```

OUTPUT

```
Enter Length of Rectangle : 2.5
Enter Breadth of Rectangle : 4.1
Area : 10.25 sq units
Enter Radius of Circle : 2.3
Area : 16.619025137490002 sq units
```

Topic _____ Date _____

Q4 =

Define an interface Shape which contains a function area(). Write the implementation of the interface for circle, rectangle and square. Also write the main() to test the interface. Can we declare a variable in an interface?

➡ CODE

```
import java.util.Scanner;
interface shape {
    void area(Scanner sc);
}
class circle implements shape {
    public void area(Scanner sc) {
        System.out.print("Enter radius of circle: ");
        double r = sc.nextDouble();
        System.out.println("Area: " + Math.PI * r * r + " sq units");
    }
}
class rectangle implements shape {
    public void area(Scanner sc) {
    ~~double l = sc.nex~~ System.out.print("Enter Length of rectangle: ");
    double l = sc.nextDouble();
    System.out.print("Enter ~~width of~~ breadth of rectangle: ");
    double b = sc.nextDouble();
    System.out.println("Area: " + l*b + " sq units");
    }
}
```

Topic _____ Date _____

```
class square implements shape {
    public void area(Scanner sc) {
        System.out.print("Enter edge length of square: ");
        double s = sc.nextDouble();
        System.out.println("Area: " + s*s + " sq units");
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        circle c = new circle();
        c.area(sc);
        rectangle r = new rectangle();
        r.area(sc);
        square s = new square();
        s.area(sc);
        sc.close();
    }
}
```

OUTPUT

```
Enter radius of circle: 3
Area: 28.274333882308138 sq units
Enter length of rectangle: 3
Enter breadth of rectangle: 4
Area: 12.0 sq units
Enter edge length of square: 5
Area: 25 sq units
```

Topic _____ Date _____

- Yes, variables can very well be declared inside of interface declarations. All variables declared inside interfaces are implicitly 'public', 'static' and 'final'. It means that variables must be initialized in the interface declaration itself. These variables essentially behave as public constants.

- Example :

```
interface Test {
    int num = 1;
    void func();
}
```

Q5 Can interfaces have constructors?

- All methods in interfaces are public and abstract. Furthermore, all data members are static, public and final and are initialized inline.

Constructors cannot be abstract and being an initializer method, it cannot initialize the data members which are static and final.

Hence, interfaces cannot have constructors.

Assignment (Week 3)

Q1. What will be the output (along with explanation) of the following programs?

(a)
```
public class JavaHungry {
    public static void main (String[] args) {
        try {
            System.out.print("A ");
            int num = 99/0;
            System.out.print("B");
        } catch (ArithmeticException ex) {
            System.out.print("C");
        } catch (Exception ex) {
            System.out.print("D");
        }
        System.out.print("E");
    }
}
```

(a) OUTPUT

```
> java JavaHungry
ACE
```

EXPLANATION

- The output of the given program is "ACE".
- The execution starts with the main() function, and jumps to the try block and executes the first statement, resulting in the first character of the output being 'A'.
- An instance of ArithmeticException is thrown in the try block due to division by zero. The execution skips to the first catch block.
- The class of exception in the first catch block matches that of the exception thrown. Hence, code in the first catch block is executed. This results in the second character of the output being 'C'.
- The execution breaks out of the try-catch blocks as all exceptions have been caught and dealt with.
- The execution of the remaining code in main() takes place and results in final character of the output being 'E'.

(b)
```
public class JavaHungry {
    public static void main(String[] args) {
        try {
            System.out.print("A");
            int num = 99/0;
            System.out.print("B");
        } catch (ArithmeticException ex) {
            System.out.print("C");
```

```
        } catch (Exception ex) {
            System.out.print("D");
        } finally {
            System.out.print("E");
        }
        System.out.print("F");
    }
}
```

(b) OUTPUT

```
>java JavaHungry
ACEF
```

EXPLANATION

- The output of the program is "ACEF".
- The execution starts at main() and jumps into the try block and executes the first statement resulting in the first character of the output being 'A'.
- An instance of ArithmeticException is thrown in the try block due to division by zero. The execution skips to the first catch block.
- The type of exception in the first catch block matches that of the exception thrown. Hence, the code in the first catch block gets executed. This results in the second character of the output being 'C'.
- As all thrown exceptions are caught, execution jumps to the finally block. This results in the penultimate character of the output being 'E'.
- The execution of the rendining code in main() takes place and results in the last character of the output being 'F'.

Q2 Create an Exception subclass UnderAge, which prints "Under Age" along with the age value when an object of UnderAge class is printed, in the catch statement. Write a class ExceptionDemo in which the method test() throws UnderAge exception if the variable age passed to it as argument is less than 18. Write main() method also to show working of the program.

Ans

```
/**** UnderAge.java ****/
public class UnderAge extends Exception {
    final private int age;
    public UnderAge(int age) {
        this.age = age;
    }
    @Override
    public String getMessage() {
        return "Under Age: " + age + " is less than 18";
    }
}
```

```
/**** exceptionDemo.java ****/
import java.util.Scanner;
class exceptionDemo {
   static void test(int age) throws UnderAge {
      if (age < 18)
         throw new UnderAge(age);
   }
   public static void main(String[] args){
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter Age: ");
      int age = sc.nextInt();
      try {
         test(age);
         System.out.println("Test successful");
      } catch (UnderAge e){
         System.out.println(e.getMessage());
         System.out.println("Test unsuccessful");
      } finally {
         sc.close();
      }
   }
}
```

OUTPUT

```
>java exceptionDemo
Enter Age: 16
Under Age: 16 is less than 18
Test Unsuccessful

>java exceptionDemo
Enter Age: 19
Test Successful.
```

Q3. Write a program to implement stack. Use exception handling to manage underflow and overflow conditions.

Ans.

```
/**** StackException.java ****/
public class StackException extends Exception {
   final private String message;
   public StackException(String message) {
      this.message = message;
   }
   @Override
   public String getMessage() {
      return this.message;
   }
}
```

```
/**** Stack.java ****/
public class Stack {
  private int tos;
  private int[] array;
  private int size;
  public Stack(int size) {
    this.tos = -1;
    this.size = size;
    this.array = new int[this.size];
  }
  public void push(int e) throws StackException{
    if (tos == size - 1)
      throw new StackException("Stack Overflow");
    else
      this.array[++this.tos] = e;
  }
  public int pop() throws StackException {
    if (this.tos < 0)
      throw new StackException("Stack Underflow");
    else
      return this.array[this.tos--];
  }
  public int getTOS(){
    return this.tos;
  }
  @Override
  public String toString(){
  return "Stack<size=" + this.size + ">";
  }
}
```

```
/**** Main.java ****/
import java.util.Random;
public class Main {
  public static void main(String[] args) {
    int r;
    Stack stack = new Stack(10);
    Random random = new Random(1337);
    System.out.println("Pushing integers onto stack...");
    while(true) {
      r = Random.nextInt(100);
      System.out.println("Pushing " + r + "...");
      try{
        stack.push(r);
        System.out.println("Elements of Stack = "
                           + (stack.getTOS() + 1));
      } catch (StackException e) {
```

```
            System.err.println(e.getMessage());
            break;
        }
    }
    System.out.println("Popping integers from stack...");
    while(true) {
        System.out.println("Elements of stack = "
                           + (stack.getTOS()+1));
        try{
            System.out.println("Popped " + stack.pop() + "...");
        } catch (StackException e) {
            System.err.println(e.getMessage());
            break;
        }
    }
  }
}
```

OUTPUT

```
> java Main
Pushing integers onto stack...
Pushing 21...
Elements of stack = 1
Pushing 44...
Elements of stack = 2
Pushing 59...
Elements of stack = 3
Pushing 22...
Elements of stack = 4
Pushing 9...
Elements of stack = 5
Pushing 48...
Elements of stack = 6
Pushing 3...
Elements of stack = 7
Pushing 4...
Elements of stack = 8
Pushing 27...
Elements of stack = 9
Pushing 67...
Elements of stack = 10
Pushing 6...
Stack Overflow
Popping integers from stack...
Elements of stack = 10
Popped 67...
Elements of stack = 9
Popped 27...
```

```
Elements of stack = 8
Popped 4...
Elements of stack = 7
Popped 3...
Elements of stack = 6
Popped 48...
Elements of stack = 5
Popped 9...
Elements of stack = 4
Popped 22...
Elements of stack = 3
Popped 59...
Elements of stack = 2
Popped 44...
Elements of stack = 1
Popped 21...
Elements of stack = 0
Stack Underflow
```

Q4. Can we write only try block without catch and finally blocks?

Ans. No, it shall result in a compilation error as follows.

|| > error : 'try' without 'catch', ~~or~~ 'finally' or resource declarations.

- The try block must be followed by either a catch or finally block.
- We can omit either one of the catch or finally blocks but not both of them.
- An exception can be made in the case of try-with-resources block which does not need to be followed by the catch or finally blocks.
- Examples of valid code:

→
```
try {
...
} catch (Exception e) {
...
}
```

→
```
try {
...
} finally {
...
}
```

→
```
try {
...
} catch (Exception e) {
...
} finally { .. }
```

→
```
try (Scanner sc = new Scanner(System.in))
{
...
}
```

Qs There are three statements in a try block: statement1, statement2, and statement3. After that there is a catch block to catch the exceptions occurred in the try block. Assume that exception has occurred in statement 2, does statement3 get executed or not?

Ans The statements preceding the one which throws the exception be executed as-is. In case that statement2 throws an exception in the try block, the execution skips to the catch block.
Hence, statement3 will not be executed.

Example:

```
public class Main{
  public static void main(String[] args){
      try{
         System.out.println(1);
         System.out.println(2/0);
         System.out.println(3);
      } catch (Exception e){
         System.out.println ("Caught");
      }
  }
}
```

OUTPUT

```
1
Caught
```

Assignment (Week 4)

Q1. What will be the output (along with explanation) of the given program?

```
import java.io.*;
class CharArrayInput {
    public static void main(String[] args) {
        String obj = "abcdef";
        int length = obj.length();
        char c[] = new char[length];
        obj.getChars(0, length, c, 0);
        CharArrayReader input1 = new CharArrayReader(c);
        CharArrayReader input2 = new CharArrayReader(c, 0, 3);
        int i;
        try {
            while((i = input2.read()) != -1) {
                System.out.print((char)i);
            }
        } catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

Ans. OUTPUT

abc

EXPLANATION

- A String ~~objected~~ was instantiated with the initial value "abcdef".
- A character array is dynamically allocated memory to contain as many characters as obj has. The characters are copied from the String object to the character array.
- An instance of CharArrayReader instantiated with the first three characters as its character array.
- A pretest loop checks whether the end of the character array is reached or not and characters are printed one by one to the console as read by the CharArrayReader instance.
- As there are only 3 characters in the instance's character array, the output consists of 'a', 'b', and 'c'.
- Hence, first three characters of "abcdef" were printed to the console.

Q2. Write a program that copies content of one file to another. Pass the names of the files through command-line arguments.

Ans.

```
/**** src.txt ****/
Sudipto Ghosh
=========
University of Delhi
```

```
/**** Copy.java ****/
import java.io.*;
public class Copy{
 public static void main (String[] args) throws Exception{
   if (args.length != 2) {
      System.err.println("Usage: java Copy <src> <dest>");
   } else {
      int i;
      FileInputStream fin = new FileInputStream(args[0]);
      FileOutputStream fout = new FileOutputStream(args[1]);
      while((i= fin.read()) != -1) {
         fout.write(i);
      }
      fin.close();
      fout.close();
      System.out.println("copied contents of " + args[0]
                          + " to " + args[1]);

   }
  }
}
```

OUTPUT

```
> java Copy src.txt dest.txt
copied contents of src.txt to dest.txt

> cat dest.txt
Sudipto Ghosh
====  ====
University of Delhi
```

Q3. Write a program to read a file and display only those lines that have the first two characters as "//" using try-with-resources.

Ans.

```
/**** LexAnalyzer.java ****/
import java.io.*;
// Sudipto Ghosh
public class LexAnalyzer {
   static boolean analyze (String line){
      if (line.length() >= 2)
         return line.substring(0, 2).equals("//");
      return false;
   }
   public static void main (String[] args) {
      if (args.length != 1) {
         System.err.println("Usage: java LexAnalyzer <file>");
      } else {
```

```
try(BufferedReader br = new BufferedReader(
                        new FileReader(args[0])) {
    String str;
    while((str=br.readLine()) != null) {
        str = str.trim();
        if(analyze(str)) {
            System.out.println(str);
        }
    }
    br.close();
} catch (Exception e) {
    System.err.println(e.getMessage());
}
    }
  }
}
// This is a line of comment
```

OUTPUT

```
>java LexAnalyzer LexAnalyzer.java
// Sudipto Ghosh
// This is a line of comment
```

Q4. How do you handle console output using PrintWriter class?

Ans. In order to handle console output using the PrintWriter class, one has to instantiate an object of the class. This class does not contain methods for writing raw bytes, and can only handle character-output streams. It prints formatted representations to a text output stream.

- The PrintWriter class has PrintWriter(OutputStream outputStream, boolean autoFlush) as one of its constructors. Here, outputStream is an object of class OutputStream and autoFlush is a boolean value which controls the automatic line-flushing behaviour of the outputStream. If autoFlush is true, outputStream is flushed whenever print, printf or println methods are invoked. Otherwise, an invocation of the flush method is required to actually flush the stream.
- Example:

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

```
        p = new PrintWriter (System.out);
        p.println("Hi");
        p.flush();

        p = new PrintWriter (System.out, true);
        p.println("Hi");
    }
}
```

OUTPUT

```
Hi
Hi
```

QS. How do you read (1) characters (2) a string, using the BufferedReader class?

Ans. Once we have instantiated BufferedReader with an instance of InputStreamReader having an InputStream, we can invoke predefined methods in the BufferedReader class to read characters and strings from the InputStream.

(a) to read characters, we can use the read() method of the BufferedReader class to read a single character as an int which can then be typecasted into a char or the read(char[] buffer, int offset, int length) method to read a number of characters into a portion of an array.

(b) to read strings, we use the readLine() method to read one line of characters at a time from the Input Stream and store it as a String object.

Example:

```
import java.io.*;
public class Main{
    public static void main(String[] args) throws IOException{
        BufferedReader br = new BufferedReader(
                                new InputStreamReader(System.in));

        String str;
        str = br.readLine();
        char[] buffer = new char[3];
        br.read(buffer, 0, 3);
        char c;
        c = (char) br.read();
        System.out.println(str);
        System.out.println(buffer);
        System.out.println(c);
        br.close();
```

```
    }
}
```

OUTPUT

```
>java Main
hello, world
hello
hello, world
hel
l
```

Q6. Write the names and meanings of all stream classes discussed in the reference book.

Ans. Input Streams in Java

(1) BufferedInputStream : contains methods to read bytes from a memory buffer.
(2) ByteArrayInputStream: contains methods to read bytes from a byte array.
(3) DataInputStream: contains methods to read Java primitive data types.
(4) FileInputStream: contains methods to read bytes from a file.
(5) FilterInputStream: contains methods to read bytes from other input streams which it uses as its basic source of data.
(6) ObjectInputStream: contains methods to read objects.
(7) PipedInputStream: contains methods to read from a piped output stream.
(8) SequenceInputStream: contains methods to concatenate multiple input streams and then read from the combined stream.

OutputStreams in Java

(1) BufferedOutputStream: contains methods to write bytes into the buffer.
(2) ByteArrayOutputStream: contains methods to write bytes into a byte array.
(3) DataOutputStream: contains methods to write Java primitive data types.
(4) FileOutputStream: contains methods to write bytes to a file.
(5) FilterOutputStream: contains methods to write to other output streams.
(6) ObjectOutputStream: contains methods to write objects
(7) PipedOutputStream: contains methods to write to a piped outputstream.
(8) PrintStream: contains methods to print Java primitive data types.

Assignment (Week5)

Q1. Write a program so that copies content of one file to another. Pass the names of the files through command-line arguments.

Ans.
```
import java.io.*;
public class Copy {
  public static void main (String[ ] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: java Copy <src> <dest>");
    } else {
      int i;
      FileInputStream fin = new FileInputStream (args[0]);
      FileOutputStream fout = new FileOutputStream (args[1]);
      while ((i= fin.read()) != -1) {
        fout.write(i);
      }
      fin.close();
      fout.close();
      System.out.println ("copied contents of " + args[0]
                          +" to " + args[1]);
    }
  }
}
```

```
/***** src.txt *****/
Sudipto Ghosh
==== ====
University of Delhi
```

OUTPUT

```
>java Copy src.txt dest.txt
Copied contents of src.txt to dest.txt
>cat dest.txt
Sudipto Ghosh
==== ====
University of Delhi
```

Q2 Write a program in Java (using try-with-resources functionality) to do the following:

→ Open two files exam1.txt and exam2.txt. Accept filenames through command line arguments.
→ Exit the program if any of the two files is unable to open.
→ Append contents of exam1.txt to exam2.txt
→ Rewrite contents of exam2.txt after removing all whitespaces from the updated content w/o using builtin methods.

Ans.

```
import java.io.*;
public class Main {
  public static void main(String[] args) {
    if (args.length != 2) {
      System.out.println("Usage: java Main <file1> <file2>");
      System.exit(1);
    }
    try(BufferedReader finA = new BufferedReader(new FileReader(args[0]));
        BufferedWriter foutB = new BufferedWriter(new FileWriter(args[1],
                                                    true))) {

      String s;
      while((s = finA.readLine()) != null) {
        foutB.newLine();
        foutB.write(s);
        foutB.flush();
      }
      System.out.println("OK (1)");
    } catch (Exception e) {
      System.out.println("Could not open one of the files, exiting...");
      System.exit(-1);
    }

    try(BufferedReader finB = new BufferedReader(new FileReader(args[1]))) {
      String s, o = "";
      while((s = finB.readLine()) != null) {
        char[] array = s.toCharArray();
        for (int i=0; i<s.length(); i++)
          switch(array[i]){
            case ' ':
            case '\t':
            case '\n':
            case '\r':
              break;
            default:
              o += array[i];
              break;
          }
      }
      try (BufferedWriter foutB = new BufferedWriter(
                                    new FileWriter(args[1]))) {

        foutB.write(o);
      }
      System.out.println("OK (2)");
    } catch (Exception e) {
      System.out.println("could not open one of the files, exiting...");
      System.exit(-1);
    }
```

```
    }
}
```

```
// exam1.txt
Semester Examination
        Sudipto
    Ghosh.

// exam2.txt
University of Delhi
```

OUTPUT

```
> java Main exam1.txt exam2.txt
OK (1)
OK (2)

> cat exam2.txt
University of DelhiSemester Examination Sudipto Ghosh.
```

Q3. What is the Delegation Event Model? Explain its components in short.

Ans. The Delegation Event Model defines standard and consistent mechanisms to generate and process events. In this model, a source generates an event and sends it to one or more listeners. The listener simply waits until it receives an event. Once an event is received, the listener processes the event and then returns. The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. Therefore, a user interface element is able to 'delegate' the processing of an event to a separate piece of code.

The components of the Delegation Event Model are as follows:

(i) Events: An event is an object that describes a state change in a source. An event can be generated as a consequence of a person interacting with the elements in a graphical user interface. Events may also occur that are not directly caused by interactions with a user interface.

(ii) Event Sources: An event source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. When an event occurs, all registered listeners are notified and receive a copy of the event object.

(iii) Event Listeners: An event listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more

sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications.

Q4. Write commonly used Event Classes in java.awt.event and their description.

Ans. Some commonly used event classes are:

(1) ActionEvent: Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.

(2) AdjustmentEvent: Generated when a scroll bar is manipulated.

(3) ComponentEvent: Generated when a component is hidden, moved, resized or becomes visible.

(4) ContainerEvent: Generated when a component is added to or removed from a container.

(5) FocusEvent: Generated when a component gains or loses keyboard focus.

(6) InputEvent: Abstract superclass for all component input event classes.

(7) ItemEvent: Generated when a check box or list item is clicked. It also occurs when a choice selection is made or a checkable menu item is selected or deselected.

(8) KeyEvent: Generated when input is received from the keyboard.

(9) MouseEvent: Generated when the mouse is dragged, moved, clicked, pressed or released. It also occurs when mouse enters or exits a component.

(10) MouseWheelEvent: Generated when the mouse wheel is moved.

(11) TextEvent: Generated when the value of a text area or text field is changed.

(12) WindowEvent: Generated when a window is activated, closed, deactivated, deiconified, iconified, opened or quit.

Assignment (Week 6)

Q1. Write different constants and their description available in AdjustmentEvent class.

Ans Some constants in the AdjustmentEvent class:

- BLOCK_DECREMENT : The user clicked inside the scroll bar to decrease its value.
- BLOCK_INCREMENT: The user clicked inside the scroll bar to increase its value.
- TRACK : The slider was dragged.
- UNIT_DECREMENT: The button at the end of the scroll bar was clicked to decrease its value.
- UNIT_INCREMENT: The button at the end of the scroll bar was clicked to increase its value.
- ADJUSTMENT_VALUE_CHANGED: A change has occurred.

Q2. Write different constants and their description available in the ComponentEvent class.

Ans. Some constants in the ComponentEvent class:

- COMPONENT_HIDDEN : The component was hidden.
- COMPONENT_MOVED : The component was moved.
- COMPONENT_RESIZED: The component was resized.
- COMPONENT_SHOWN: The component became visible.

Q3. Explain syntax of all constructors available in ContainerEvent and FocusEvent Class.

Ans. Constructors in ContainerEvent Class are:

- ContainerEvent (Component src, int type, Component comp)
  - → It instantiates a ContainerEvent Object.
  - → The first argument src, is a reference to the component object (container) that originated the event. This constructor throws an IllegalArgumentException if src is null.
  - → The second argument type, is an integer indicating the type of the event.
  - → The third argument comp is the reference to the Component that was added or removed.

Constructors in FocusEvent class are:

- FocusEvent (Component src, int type)
  - → It instantiates a FocusEvent object and identifies it as a permanent change in focus.
  - → The first argument src is the Component that originated the event. This constructor throws an IllegalArgumentException if src is null.
  - → The second argument type is an integer indicating the type of the event.

- FocusEvent (Component src, int type, boolean temporaryFlag)
  - → It instantiates a FocusEvent object and identifies whether or not the change is temporary.
  - → The first argument src is the component that originated the event. This constructor throws an IllegalArgumentException if src is null.
  - → The second argument type, is an integer indicating the type of the event.
  - → The third argument temporaryFlag is a boolean value which is set to true if the focus change is temporary, and false if it is permanent.

- FocusEvent (Component src, int type, boolean temporaryFlag, Component other)
  - → It instantiate a FocusEvent object with the specified temporary state and an opposite component. The opposite component is the other Component involved in the focus change. for a FOCUS_GAINED event, this is the component that lost focus. for a FOCUS_LOST event, this is the component that gained focus. If the focus change occurs with a native application, with a Java application in a different VM or with no opposite Component, then the other component is null.
  - → The first argument src, is the component that originated the event. This constructor throws an IllegalArgumentExcepton if src is null.
  - → The second argument type, is an integer indicating the type of the event.
  - → The third argument temporaryFlag is a boolean value which is set to true if the focus change is temporary and false if it is permanent.
  - → The fourth argument other is the opposite component involved in the focus change or null.

Q4 Write the name of constants present in InputEvent class.

Ans Constants present in InputEvent class:

- ALT_MASK
- BUTTON2_MASK
- META_MASK
- ALT_GRAPH_MASK
- BUTTON3_MASK
- CTRL_DOWN_MASK
- SHIFT_MASK
- BUTTON1_MASK
- CTRL_MASK
- ALT_DOWN_MASK
- BUTTON2_DOWN_MASK
- META_DOWN_MASK
- ALT_GRAPH_DOWN_MASK
- BUTTON3_DOWN_MASK
- SHIFT_DOWN_MASK
- BUTTON1_DOWN_MASK

Q5 List all constants present in KeyEvent class.

Ans Constants present in KeyEvent class:

- KEY_PRESSED
- KEY_RELASED
- KEY_TYPED
- VK_0 to VK_9
- VK_A to VK_Z
- VK_ALT
- VK_DOWN
- VK_LEFT
- VK_RIGHT
- CHAR_UNDEFINED
- VK_CANCEL
- VK_ENTER
- VK_PAGE_DOWN
- VK_SHIFT
- VK_CONTROL
- VK_ESCAPE
- VK_PAGE_UP
- VK_UP
- VK_UNDEFINED

Q6 List all constants present in MouseEvent class.

Ans Constants present in MouseEvent class:

- MOUSE_CLICKED
- MOUSE_DRAGGED
- MOUSE_ENTERED
- MOUSE_EXITED
- MOUSE_MOVED
- MOUSE_PRESSED
- MOUSE_RELEASED
- MOUSE_WHEEL

Q7. List all constants and their meaning present in WindowEvent class.

Ans Constants in WindowEvent class:

- WINDOW_ACTIVATED: The window was activated.
- WINDOW_CLOSED: The window has been closed.
- WINDOW_CLOSING: The user has requested that the window be closed.
- WINDOW_DEACTIVATED: The window was deactivated.
- WINDOW_DEICONIFIED: The window was deiconified.
- WINDOW_GAINED_FOCUS: The window has gained input focus.
- WINDOW_ICONIFIED: The window was iconified.
- WINDOW_LOST_FOCUS: The window lost input focus.
- WINDOW_OPENED: The window was opened.
- WINDOW_STATE_CHANGED: The state of the window changed.