OOP Code

2020-21

1 (b)

i) Error: The method h() is defined inside the method g(), which is not allowed in Java.

Correction: Move the definition of h() outside the g() method.

ii) Error: The method product() is declared as void, which means it cannot return a value. However, there is a return result; statement at the end.

Correction: change the return type of Product() from void to int.

iii) Error: The sum method computes the sum but does not return the result. Since sum has a return type of int, it must have a return statement.

correction: Add return result; at the end of the statement.

iv) Error: The variable a is declared twice, once as a parameter in the method signature and once inside the method body.

correction: Remove the second declaration of a in the method body.

— 2 —

3(a)

```
class AddAmount{
    Private int amount=50;
  Public AddAmount(){
        }
  Public AddAmount(int money){
      amount += money;
    }
  Public void displayAmount(){
  System.out.PrintIn("Amount is : " + amount);
  }
}
Public class Main{
  Public static void main(String[] args){
  AddAmount piggyBank1 = new AddAmount();
   PiggyBank1.displayAmount();
```

```
    AddAmount piggyBank2= new AddAmount(30);
    PiggyBank2.displayAmount( );
  }
}
```

Output:

```
Amount is : 50
Amount is : 80
```

4(b)

```
abstract class Animal {
    Private String name;
    Public void setName(String name){
     this.name = name;
    }
    Public String getName( ) {
        Return name;
     }
    Public abstract void eat( );
}

class Dog extends Animal{
     @Override
      Public void eat( ) {
     System.out.println("Dog is Eating");
     }
}
```

```
Public class Main {
    Public static void main(String[] args) {
        Dog dog = new Dog();
        dog.setName("Zoom");
        System.out.println("Dog's name : " +
                dog.getName());
        dog.eat();
    }
}
```

Output:

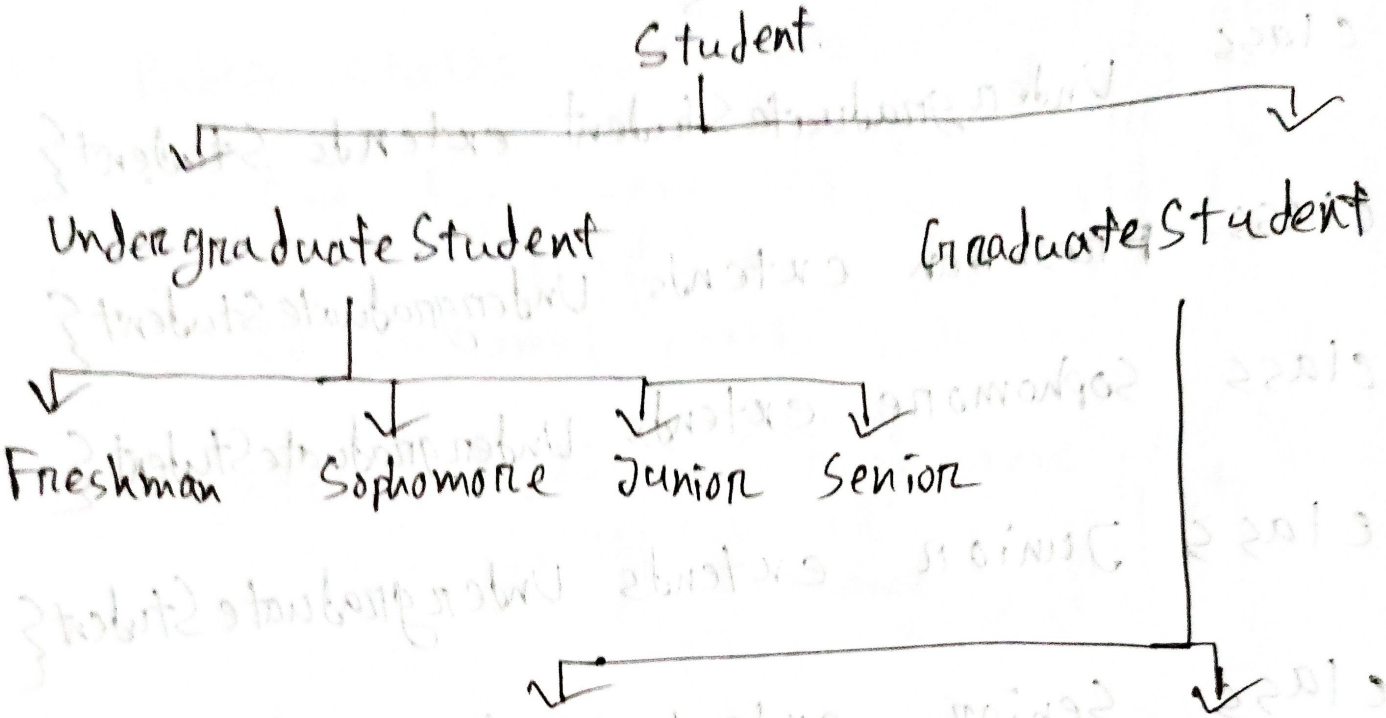
```
Dog's name : Zoom
Dog is Eating
```

—x—

4(c)

```
class Student{ }
class UndergraduateStudent extends Student{ }
class Freshman extends UndergradciateStudent{ }
class Sophomore extends UndergraduateStudent{ }
class Junior extends UndergraduateStudent{ }
class Senior extends UndergraduateStudent{ }
class GraduateStudent extends Student { }
class MasterStudent extends GraduateStudent{
}
class DoctoralStudent extends
                    GraduateStudent {
}
```

Inheritance Hierarchy:



Relationships:

Here, Student is the base class representing a general student of the university. UndergraduateStudent and GraduateStudent extend student, differentiating students based on their academic level. also undergraduateStudent has subclasses, Freshman, sophomore, Junior and senior. GraduateStudent has subclasses, mastersStudent and DoctoralStudent.

5(c)

```
import java.util.Scanner;

public class StudentMarks {

    public static void main(String[] args){

    Scanner scanner= new Scanner(System.in);

    int rollNO1 = Scanner.nextInt();

    int subject1Marks1 = Scanner.nextInt();

    int Subject2Marks1 = Scanner.nextInt();

    int totalMarks1 = Subject1Marks1 + Subject2Marks1;

    int rollNO2 = Scanner.nextInt();

    int Subject1Marks2 = scanner.nextInt();

    int Subject2Marks2 = Scanner.nextInt();

    int totalMarks2 = Subject1Marks2 + Subject2Marks2;
```

```
int highestMark1, highestMark2;

System.out.println("Roll No: "+rollNo1+
        "- Total Marks: "+totalMarks1);
System.out.println("Roll No: "+rollNo2+
        "- Total Marks: "+totalMarks2);

if(subject1Marks1 > subject1Marks2){
    highestMark1 = subject1Mark11;
System.out.println("Subject 1 Highest: "+
    highestMark1+"-Roll No: "+rollNo1);
}
else{ highestMark1 = subjectMarks2;
System.out.println("Subject 1 Highest: "+
    highestMark1+"-Roll No: "+rollNo2);}
```

```
if (Subject1Marks1 > Subject2Marks2) {
    highestMark2 = Subject2Marks1;
System.out.println("Subject 2 Highest : " +
    highestMark2 + "-Roll NO : " + rollNO1);
}
else { highestMark2 = Subject2Marks2;
System.out.println("Subject 2. Highest : " +
    highestMark2 + "Roll NO : " + rollNO2);
}
}
}
```

Input:

```
1  70  80
2  60  90
```

Output:

```
Roll NO : 1 - Total Marks : 150
Roll NO : 2 - Total Marks : 150
Subject 1 Highest 70 - Roll NO : 1
Subject 2 Highest 90 - Roll NO : 2
```

6 (a)

i) Output: 7 1

Explanation: here the mod result variable is storing the value 7 after operation i%J and the variable divresult is storing the value of i/mod result which is 1.

ii) Output: OK

Explanation: here, the indices of array i has different values. so, 1st and 2nd indices is not same and variable J stores the value 1. In the if statement J is greater than or equal to 1, so OK prints.

6(b)

```
class ExceptionA extends Exception {
    Public ExceptionA(String message){
        Super(message);
    }
}

class ExceptionB extends ExceptionA {
    Public ExceptionB(String message){
        Super(message);
    }
}

class ExceptionC extends ExceptionB {
    Public ExceptionC(String message){
        Super(message);
    }
}
```

```
Public class ExceptionDemo {
    Public static void main(String [] args){
        try{
        throw new exceptionC("this is ExceptionC");
        }
    catch(ExceptionA e){
        System.out.println("caught: "+e.getMessage());
        }
    }
}
```

6(c)

```
Package test;

public class MinMax{
  Public static void main(String [ ] args){
  int [ ] numbers = {10,0, 20, 30,5, 21, 30};
   int max_num = numbers[0];
   int min_num = numbers[0];

   for(int num : numbers){
      if(num > max_num) max_num = num;
      if(num < min_num) min_num = num;
   }
    System.out.println("Largest value : "max_num);
    System.out.println("Smallest value: "min_num);
  }
}
```

7 (a)

```
class TaskA extends Thread {
    Public void run( ) {
        for(int i=0; i<100; i++) {
        System.out.println("Hello world");
        }
    }
}

class TaskB extends Thread {
    Public void run( ) {
        int sum=0;
    for(int i=1; i<=1000; i++) {
        sum+=i; }
    }
}
```

```
class TaskC extends Thread {
    Public void run() {
    for(int i=0; i<200; i++) {
        system.out.println("I can do it");
        Thread.sleep(1000);
    }
    }
}

Public class TaskManager {
    Public static void main(String[] args) {
        Thread taskA = new TaskA();
        Thread taskB = new TaskB();
        Thread taskC = new TaskC();
        taskC.start();
        taskC.join();
        taskA.start();
        taskA.join();
        taskB.start();
    }
}
```

2019-20

1(b)

i) Output: Compilation error.

Explanation: In Java, if statement expects a boolean expression, but -1 is an integer, not a boolean. Also the break statement is being used outside loop or switch statement.

ii) Output: CSE

Explanation: This code simply concatenates three strings "C", "S", and "E" using the + operator and prints them.

iii) Output: Compilation Error

Explanation: The for loop's condition 1 is an integer, but it is not a boolean expression, which is required in Java.

Output: No output, but the code will compile sucessfully.

Explanation: This code contains two main methods, but only the first one with the signature Public static void main(String[]arrr) is valid.

3(a)

```
Public class BankAccount {

    Private String depositorName;
    Private String accountNumber;
    private String accountType;
    Private double balance;

Public BankAccount (String n, String a, String
        t, double b) {
    depositorName = n;
    accountNumber = a;
    accountType = t;
    balance = b;
                                    }
```

```
Public void deposit(double amount) {
    if(amount>0) {
        balance += amount}
}

Public void withdraw(double amount){
    if(amount>0 && amount <=balance) {
        balance -= amount; }
}

Public void display() {
    System.out.println("Name: " + depositorName);
    System.out.println("Balance: " + balance);
}
}
```

5(c)

```
import java.util.Scanner;

Public class ElectionCount {
    Public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int [] votes = new int [5];
    int spoiltBallots = 0;
    while (Scanner.hasNextInt()) {
      int ballot= scanner.nextInt();
      if( ballot >= 1 && ballot<=5) {
        votes[ballot-1]++; }
      else { spoiltBallots++; }
    Scanner.close();
    for (int i=0; i< votes.length; i++) {
      System.out.println("candidate "+(i+1)+ ": " + votes[i]+
                                    " votes");
    }
```

```
System.out.println("Spoilt ballots: " +
                   spoiltBallots);
    }
}
```

Z(b)

```
Public class ElevatorTest {
    Public Static void main(String[] args) {
        Elevator elevator = new Elevator();
        elevator.setState(new doorIsOpen());
        elevator.getState().doAction();
        elevator.setState(new doorIsClosed());
        elevator.getState().doAction();
    }
}
```

2018-19

1(b)

ii) Output: 418

Explanation: Here, 'j', 'a', 'v', 'a' are character literals. When characters are added with +, their ASCII values are used.

'j' = 106

'a' = 97

'v' = 118

'a' = 97

Thus summation resulting in 418

iii) Output: No output (compiles successfully without error)

Explanation: In Java $ and _ are valid characters for variable names. The variable has the value 5.

2(b)

The given code is in C# language.

Errors:

i) The keyword should be in lowercase.

ii) There should be a space before the colon in public class Honda4 : Hond.

iii) Honda4 is attempting to inherit from Hond, which is marked as sealed. In C#, a sealed class cannot be inherited.

Fix: To Fix it sealed modifier has to be removed.

iv) Bike bike = new Bike(); attempts to instantiate an abstract class, which is not allowed. Bike is abstract and cannot be instantiated directly.

3(a)

Multilevel inheritance is a type of inheritance in oop where a class is derived from another derived class, creating a chain of inheritance.

Implementing the scenario using oop:

```
class SuperClass {
    public void display() {
        System.out.println("I am super class");
    }
}

class ChildClass extends SuperClass {
    @Override
    public void display() {
        System.out.println("I am child class");
    }
}
```

3(b)

```
class Staff{
    Private int code;
    Private String name;

    Public Staff(int code, String name){
        this.code = code;
        this.name = name;
    }
    Public int getCode(){
        return code;
    }
    Public String getName(){
        return name;
    }

    Public void displayInfo(){
    System.out.println("Code: "+code+", Name : "+name);
    }

}
```

```
class Teacher extends Staff{
    private string subject;
    private string publication;
Public Teacher(int c, String n, String s,
        String p){
    super(c,n);
    subject = s;
    publication = p;
}
Public void displayInfo(){
    Super.displayInfo();
System.out.println("subject: "+ subject+"Publication:"
        + Publication);
}
}
```

```
class Typist extends Staff{
        int speed;
    Public Typist(int code, String name, in speed)
{       super(code,name);
        this.speed = speed; }
    public void displayInfo(){
        super.displayInfo();
    System.out.println("speed:" +speed + "wpm");
    }
}
class RegularTypist extends Typist {
    Public RegularTypist(int code, String name,
                int speed){
        super(code, name, speed);
    }
```

```
Public void displayInfo(){
    Super.displayInfo();
    System.out.println("Type: Regular Typist");
}
}

class CasualTypist extends Typist{
    double dailywages;
Public CasualTypist(int c, String n, int s,
    double d){
    Super(c,n,s);
    this.dailywages = d;
}
    public void displayInfo(){
        Super.displayInfo();
System.out.println("Type: casual", Daily wages
                    : " + dailywages);
}
}
```

```
class officer extends staff{
    string grade;
    Public officer(int c, String n, String g)
    {   super(c,n);
        this.grade= g;
    }
    public void displayInfo(){
        super.displayInfo();
        System.out.println("Grade: "+grade);
    }
}
```

4(b)

```
abstract class Animal {

    Protected int legs;

    Protected Animal (int legs) {
        this. legs = legs;
    }

    Public void walk() {
    System. out. println("This animal walks
    with " + legs + " legs. ");
    }

    Public abstract void eat();

}
```

5(b)

```
import java.util.Scanner;

class Student {
    int roll;
    int subject1;
    int subject2;
    int subject3;
    int totalMarks;
Student(int roll, int subject1, int subject2,
        int subject3) {
    this.roll = roll;
    this.subject1 = subject1;
    this.subject2 = subject2;
    this.subject3 = subject3;
    this.totalMarks = subject1 + subject2 +
                      subject3;
}
}
```

```
Public class StudentMarks{
  Public Static void main(String[] args){
Scanner scanner = new Scanner(System.in);
int numberOfStudents=50;
Student[] students = new Student[numberofStudents];
for(int i=0; i<50; i++){
    int roll=Scanner.nextInt();
    int subject1=Scanner.nextInt();
    int subject2=Scanner.nextInt();
    int subject3=Scanner.nextInt();
  Students[i] = new Student(roll, subject1, subject2,
                            subject3);
  }
  Scanner.close();
```

```
int h_Subj_1 =0, h_Subj_2=0, h_Subj_3=0;
int h_1_roll=0, h_2_roll=0, h_3_roll=0;
int highestTotalMarks=0;
int higest_Mark_Roll=0;
for(Student student : students){
if(student.Subject1>h_Subj_1){
h_Subj_1 = student.Subject1;
h_1_roll = student.roll;
}
if(student.Subject2 > h_Subj_2){
h_Subj_2 = student.Subject2;
h_2_roll = student.roll;}
if( student.Subject3> h_Subj_3){
h_Subj_3 = student.Subject3;
h_3_roll = student.roll;}
```

```
if(Student.totalMarks > highestTotalMarks){
    highestTotalMarks = student.totalMarks;
    highest_MARK_Roll = Student.roll;
}
}

System.out.println("Subject1:"+h_Subj_1+"Roll:"
                                    h_1_roll);
System.out.println("Subject2:"+h_Subj_2+"Roll:"
                                    h_2_roll);
System.out.println("Subject3:"+h_Subj_3+
                    "Roll:"h_3_roll);
System.out.println("High marks:"+highestTotal
    Marks+", Mark roll:"+
                    highest_Mark_Roll);
}
```

5(c)

Issues in the code that will be causing compile errors:

i) The variable dest is declared as new String(), which does not support append operations. To build a string character by character, use StringBuilder instead.

ii) The code has Palindrome.charAt(i), which should be charAt(i).

connections:

```
StringBuilder dest = new StringBuilder();
dest.append(Palindrom.charAt(i));
```

6(a)

various form of implementing interfaces;

i) A class can implement an interface by providing concrete implementations for each of the interface's methods.

```
interface Animal{
        void sound();
}
class Dog implements Animal{
    Public void sound(){
        System.out.println("woof");
    }
}
```

ii) An interface can be implemented using an anonymous inner class, which is useful when you only need a single-use implementation.

```
interface Greeting{
        void sayHello();
}

Public class Main {
    Public static void main (String[] args){
    Greeting greeting = new Greeting(){
      Public void sayHello(){
      System.out.println("Hello");
      }
    };
    greeting.sayHello();
}
}
```

iii) For functional interfaces, Java allows using lambda expressions to implement the interface.

```
interface Calculator{
        int add(int a, int b);
}

Public class Main{
    Public static void main(String[] args){
        Calculator calculator = (a,b) -> a+b;
    System.out.println(calculator.add(5,3));
    }
}
```

6(b)

The code will not compile successfully due to a small issue in the main method of the InterfaceTest class. The main method should have the parameter String[] args instead of String args[].

connected code:

```
class InterfaceTest {
Pablic Statie void main(String[] args){
 NewShape ne1 = new Circle1();
NewShape ne2 =new NewCirele2();
ne1.draw();
ne2.draw();
}
}
```

6(d)

No, If the draw() method in NewCircle is set to private, the samePackage class will not be able to access it, even if it is in the same pacange.

In Java, Private methods are only accessible within the same class.

To allow access within the same package visibility has to be changed to default, Protected or Public.

— 6 —

7(b)

The code has an issue with how the threads are started. In Java calling run() directly does not start a new thread, it simply executes the run method in the current thread. To Actually run the code in separate threads, you need to use start() instead of run().

connected code:

```
Public Static void main(String[] args){
    t1  t1 = new t1();
    t2  t2 = new t2();
    t1. set priority(Thread.MIN_PRIORITY);
    t2. set priority(Thread.MIN_PRIORITY);
    t1. start();
    t2. start();
}
}
```

7 (e)

Errors in the given code:

i) Duplicate public keyword in "public public static void main (String args[]).

ii) Syntax of the parameter in main is not correct, it should be (String [] args)

iii) (ch = fr.read()) != 1 is incorrect. It should be (ch = fr.read()) != -1.

iv) fr.write(ch) is incorrect, as fr is a FileReader object, fw.write(ch) should be used.

— e —

7(d)

```
Public class Test{
  Public static void main(String[] args){
    try{
        int result = 10/0;
        int[] numbers = {1,2,3};
        int number = numbers[5];
        String text = null;
        int length = text.length();
    }
  catch(ArithmeticException e){
    System.Out.println("can't Divide by zero");
  }
  catch(ArrayIndexOutOfBoundsException e){
  System.out.println("Array index is out of bound");
  }
```

```
catch(NullPointerException e){
  System.out.println("can't access null object");
  }

}

}
```

17-18

1 (c)

```
import java.util.Scanner;
Public class main {
  Public Static double power(double m, int n) {
      return Math.Pow(m,n);
    }
  Public Static double power(double m) {
      return power(m, 2);
  }
  Public static void main(String [] args){
    Scanner scanner = new Scanner(System.in);
    double m = scanner.nextDouble();
    System.Out.Println(" Enter the value of
      n or press enter to calculate
      square");
```

```
Scanner.nextLine();
String input = scanner.nextLine();
double result;
if (input.isEmpty()){
    result = Power(m);
}
else {
    int n = Integer.ParseInt(input);
    result = Power(m,n);
}
System.out.println("Result:" + result);
}
}
```

3(c)

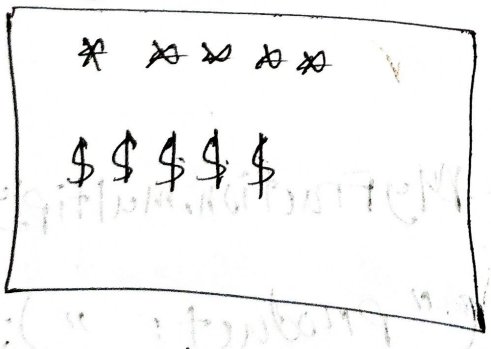
```
Public class MyFraction {
    Private int n;
    Private int d;
    Public MyFraction() {
        this.numerator = 0;
        this.denominator = 1;
    }
    Public MyFraction(int n, int d) {
        this.n = n;
        this.d = d;
    }
    Public MyFraction add(MyFraction num) {
        int newN = this.n * num.d + num.d * this.d;
        int newD = this.d * num.d;
        return new MyFraction(newN, newD);
    }
```

```
Public Static MyFraction multiply(MyFraction f1,
                                  MyFraction f2){
    int newN = f1.n * f2.n;
    int newD = f1.d * f2.d;
    return new MyFraction(newN, newD);
}
    Public void display() {
        System.out.println(n + "/" + d); }
Public Static void main(String[] args){
    MyFraction f1 = new MyFraction(1,2);
    MyFraction f2 = new MyFraction(3,4);
    MyFraction sum = fraction1.add(fraction2);
    System.out.println("sum: ");
    sum.display();
    MyFraction product = MyFraction.multiply(f1,f2);
    System.out.println("product: ");
    Product.display();
}
}
```

4(a)

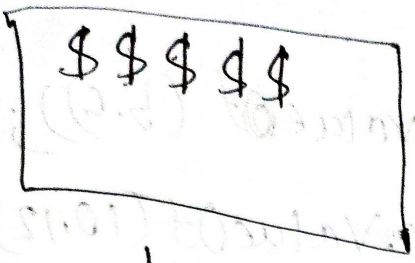
The dangling-else problem occurs in programming languages like Java when if statements are nested without using braces { } to clearly associate else statements. Without braces, it's unclear which if an else belongs to. By default, an else matches the closest unmatched if.

Output if x is 9 and y is 11:



1st condition is true, so it enters if and second also true, so it executes last print is out of condition.

Output if x is 11 and y is 9:



1st condition is false, But the last print is out of condition.

—x—

4(c)

```
import Java.util.ArrayList;
Public class FloatingPointList{
Public static void main(String[] args){
    ArrayList<Double> numbers = new ArrayList<>();
    numbers.add(12.34);
    numbers.add(34.5);
    numbers.add(5.6);
    numbers.add(7.89);
    numbers.add(10.12);
    numbers.add(3.45);
```

```
System.out.println(numbers.size());

numbers.remove(Double.valueOf(5.6));

numbers.remove(Double.valueOf(10.12));

System.out.println(numbers);

}

}
```

5(b)

Implementing the inheritance hierarchy:

```
class A { public void display() {
        System.out.println("class A");
    }
}
```

```
class B extends A {
    @Override
    public void display(){
        System.out.println("class B");
    }
}

class C extends B{
    @Override
    public void display(){
        System.out.println("class C");
    }
}

class D extends B{
    @Override
    public void display(){
        System.out.println("class D");
    }
}
```

6(c)

```
class Employee {
    Public String name;
    Public String designation;
    Public int experience;
    private String birthdate;
    Protected double salary;
    Protected String address;

Public Employee(String name, String
  designation, int experience, String birthdate,
  double Salary, String address) {
    this.name=name;
    this.designation=designation;
    this.experience = experience.
    this.birthdate=birthdate.
    this.salary = salary;
    this.address = address; }
```

```
public void displayDetails(){
    System.out.println("Name: " + name);
    System.out.println("Designation: "+designation);
    System.out.println("Experience: "+ experience+
                        " years");
    System.out.println(" Information collected ");
    displayprotectedInfo();
}
    private void displayBirthDate(){
        System.out.println("Birth Date: "+birthdate);
    }
        protected void displayprotectedInfo(){
            system.out.println("salary" +salary);
            System.out.println("Address " +Address);
        }
}
```

7(c)

```
Public class BankAccount {
    Private double balance;
  Public BankAccount(double i) {
    if( i < 0){
      throw new IllegalArgument Exception("
          Initial balance can't be negative");
    }
    this.balance = i;
}
Public void withdraw(double amount){
    if(amount<0) {
   throw new IllegalArgumentException("withdrawl
     amount cant be negative"); }
   if(amount > balance) {
throw new IllegalArgumentException("withdrawl
   amount must not exceed current balance");
}
```

```
balance -= amount;
}

public void deposit(double amount){
    if(amount<0){
throw new IllegalAmountException("Deposit
amount cant be negative");
    }
    balance += amount;
}
```

8(c)

a) Output : A2 B.2

b) Output: compilation ERROR, here, one() method in class A is being called but class B has overriden one(int x) instead of one().

c) Output: WOOOO!!

d) Output: ERROR, Blue is abstract; cannot be instantiated.

here, Blue is an interface and interface cannot be instantiated directly in java. It will result in compilation error.

e) Output: ERROR: A cannot be converted to C

here, C is a subclass of A, but A is not a subclass of C. This means that A cannot be assigned as an object to a C reference.

f) Output: cannot find symbol.

here, A is a superclass of B, and B implements both Red and Blue. But turnBlue() is not defined in A; it is defined in the B. since a is a type of A, It can only call methods defined in A.

g) Output: C.1
A.2