**OOP\_HackerRank\_PropertyInititalization\_Using**

School and copy of a School using copy constructor

Create a Java class named School with following attributes:

private String name

private String location

private Student students and

private Principal principal

Implement a parameterised constructor to initialise all the instance variable of School class.

Implement a copy constructor for School class that takes a School object and initialise all the instance variable of new School object with the same attributes.

Take a pair of setter and getter method for all the instance variables.

In the School class, create a method named addStudent(Student student) that adds a new Student object to the students .

Create a Student class and this class should have following attributes:

private String name,

private int age,

private double GPA and a

private Teacher object named teacher.

Implement a parameterised constructor to initialise all the instance variable of Student class.

Take a pair of setter and getter method for all the instance variables.

In the Student class, create a method named changeTeacher(Teacher newTeacher) that changes the teacher of the student to the specified teacher.

Create a Teacher class and this class should have following attributes:

private String name and

private String subject.

Implement a parameterised constructor to initialise all the instance variable of Teacher class.

Take a pair of setter and getter method for all the instance variables.

Create a Principal class this class should have following attributes:

private String name and

private int yearsExperience.

Implement a parameterised constructor to initialise all the instance variable of Principal class.

Take a pair of setter and getter method for all the instance variables.

Create a Main class that create a School object and sets its attributes and the attributes of its students, teacher, and principal. Then, create a copy of the School object using the copy constructor and print the attributes of both the original and the copy of the School objects to confirm that the copy constructor works correctly.

Finally, use the addStudent() method to add a new student to the original School object and use the changeTeacher() method to change the teacher of one of the original School object's students. Print the attributes of both the original and the copy of the School objects again to confirm that the copy constructor created a deep copy of the Student, Teacher, and Principal objects and that the addStudent() and changeTeacher() methods did not affect the students or teachers of the copy of the School object.

//-------------------------------school

**class** School

{

**private** String name;

**private** String location;

**private** Student students;

**private** Principal principal;

**public** School(String name, String location, Student students, Principal principal)

{

**this**.name = name;

**this**.location = location;

**this**.students = students;

**this**.principal = principal;

}

**public** School (School s)

{

**this**.name = s.name;

**this**.location = s.location;

**this**.students = s.students;

**this**.principal = s.principal;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getLocation() {

**return** location;

}

**public** **void** setLocation(String location) {

**this**.location = location;

}

**public** Student getStudents() {

**return** students;

}

**public** **void** setStudents(Student students) {

**this**.students = students;

}

**public** Principal getPrincipal() {

**return** principal;

}

**public** **void** setPrincipal(Principal principal) {

**this**.principal = principal;

}

@Override

**public** String toString() {

**return** "School [name=" + name + ", location=" + location + ", students=" + students + ", principal=" + principal

+ "]";

}

}

//-----------------------------Student

**class** Student

{

**private** String name;

**private** **int** age;

**private** **double** GPA;

**private** Teacher teacher;

@Override

**public** String toString() {

**return** "Student [name=" + name + ", age=" + age + ", GPA=" + GPA + ", teacher=" + teacher + "]";

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** **double** getGPA() {

**return** GPA;

}

**public** **void** setGPA(**double** gPA) {

GPA = gPA;

}

**public** Teacher getTeacher() {

**return** teacher;

}

**public** **void** setTeacher(Teacher teacher) {

**this**.teacher = teacher;

}

**public** Student(String name, **int** age, **double** gPA, Teacher teacher)

{

**this**.name = name;

**this**.age = age;

GPA = gPA;

**this**.teacher = teacher;

}

**void** changeTeacher(Teacher newTeacher)

{

**this**.teacher=newTeacher;

}

}

//-----------------------------------teacher

**class** Teacher

{

**private** String name ;

**private** String subject;

**public** Teacher(String name, String subject) {

**super**();

**this**.name = name;

**this**.subject = subject;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getSubject() {

**return** subject;

}

**public** **void** setSubject(String subject) {

**this**.subject = subject;

}

@Override

**public** String toString() {

**return** "Teacher [name=" + name + ", subject=" + subject + "]";

}

}

//---------------------------principal

**class** Principal

{

**private** String name ;

**private** **int** yearsExperience;

**public** Principal(String name, **int** yearsExperience) {

**super**();

**this**.name = name;

**this**.yearsExperience = yearsExperience;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getYearsExperience() {

**return** yearsExperience;

}

**public** **void** setYearsExperience(**int** yearsExperience) {

**this**.yearsExperience = yearsExperience;

}

@Override

**public** String toString() {

**return** "Principal [name=" + name + ", yearsExperience=" + yearsExperience + "]";

}

}

**public** **class** MyProgram

{

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

{

Teacher tec=**new** Teacher("james ", "Core java");

Student stu=**new** Student("Sourab das", 20, 98.2, tec);

Principal pri=**new** Principal("Rechi", 20);

School sch=**new** School("Monessori high school" , "Odisa", stu, pri);

School cpysch=**new** School(sch);

System.***out***.println(sch);

System.***out***.println(cpysch);

Student newStu=**new** Student("Bajrang singh", 22, 99.0, tec);

Teacher chTec=**new** Teacher("Natraj ", "Core java");

newStu.changeTeacher(chTec); //-----------change teacher for new student does not effect copy school

sch.setStudents(newStu);//-----------------add student

System.***out***.println(sch);

System.***out***.println(cpysch);//-------did not affect the students or teachers of the copy of the School object

}

/\*copy of the School objects again to confirm that the copy constructor created a deep copy of the Student,

Teacher, and Principal objects and that the addStudent() and changeTeacher()

methods did not affect the students or teachers of the copy of the School object.\*/

}

**JB\_OOP\_Constructor\_And\_Methods**

**Define a class Temperature described below:**

**Class name : Temperature**

**Data members/Instance variables:**

**double max to store maximum temperature**

**double min to store minimum temperature**

**Member Methods:**

**i. A parameterised constructor to input the maximum (max) and minimum (min) temperature of a day in Celsius.**

**ii. To compute the maximum and minimum temperatures in Fahrenheit**

**as:**

**°F = °C × (9/5) + 32**

**iii. To display the maximum and minimum temperatures in Fahrenheit.**

**Write a main method to create an object of a class and call the above member methods.**

**class** Temperature

{

**double** max;

**double** min;

Temperature(**double** max,**double** min)

{

**this**.max=max;

**this**.min=min;

}

**void** calFar()

{

System.***out***.println("maximmum temperature in Fahrenheit :"+(max+(9/5)+32));

System.***out***.println("minimum temperature in Fahrenheit :"+(min+(9/5)+32));

}

}

**public** **class** MyProgram

{

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

{

Temperature t=**new** Temperature(100,65);

t.calFar();

}

}

**OUTPUT:**

maximmum temperature in Fahrenheit :133.0

minimum temperature in Fahrenheit :98.0

Define a class Marks as per the given specifications:

Class name Marks

Data members/Instance variables:

name, age, m1, m2, m3, (marks in three subjects), maximum, average

Member Methods:

i. A parameterised constructor to input the details of a student viz. name, age, m1, m2 and m3 (marks in three subjects)

ii. To compute the average and the maximum out of three marks

iii. To display the name, age, marks in three subjects, maximum marks and average.

Write a main method to create an object of a class and call the above member methods.

**class** Marks

{

String name;

**int** age;

**int** m1,m2,m3,max;

**double** avg;

Marks(String name,**int** age,**int** m1,**int** m2,**int** m3)

{

**this**.name=name;

**this**.age=age;

**this**.m1=m1;

**this**.m2=m2;

**this**.m3=m3;

}

**void** calMaxNavg()

{

**this**.avg=(**double**)(m1+m2+m3)/3.0;

**this**.max=((m1>m2&&m1>m3)?m1:(m2>m1&&m2>m3)?m2:m3);

System.***out***.println("Name of the student :"+name);

System.***out***.println("Age of the student :"+age);

System.***out***.println("Mark in m1 :"+m1);

System.***out***.println("Mark in m2 :"+m2);

System.***out***.println("Mark in m3 :"+m3);

System.***out***.println("Maximun :"+max);

System.***out***.println("Average :"+avg);

}

}

**public** **class** MyProgram

{

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

{

Marks m=**new** Marks("Bhawesh",20,50,80,75);

m.calMaxNavg();

}

}

**OUTPUT:**

Name of the student :Bhawesh

Age of the student :20

Mark in m1 :50

Mark in m2 :80

Mark in m3 :75

Maximun :80

Average :68.33333333333333

**JB\_OOP\_**

# MCQ

1. **Correct**

Question: 1

What is Inheritance in Java programming?

* 1. 

a. It’s a Java-specific term for importing packages.

* 1. 

b. It’s a process of creating a new class using the main() method.

* 1. 

c. It’s a technique to create objects in Java.

* 1. Correct Answer



d. It’s a process where one class acquires the properties (fields)  
and behaviors (methods) of another class.

Answered

1. **Correct**

Question: 2

**\_\_\_\_\_\_\_** keyword is used for inheritance in Java.

* 1. 

a. class

* 1. 

b. new

* 1. Correct Answer



c. extends

* 1. 

d. this

Answered

1. **Correct**

Question: 3

Can a subclass inherit private members of its superclass?

* 1. 

a. Yes

* 1. Correct Answer



b. No

Answered

1. **Incorrect**

Question: 4

class Animal {  
public void move() {  
System.out.println(“The animal moves”);  
}  
}

class Cheetah Extends Animal {  
public void move() {  
System.out.println(“The cheetah moves swiftly”);  
}  
}  
public class Main {  
public static void main(String[] args) {  
Cheetah cheetah = new Cheetah();  
cheetah.move();  
}  
}

* 1. 

a. The animal moves

* 1. Incorrect Answer



b. The cheetah moves

* 1. Correct Answer



c. Compile Time error

* 1. 

d. None of the above

Answered

1. **Incorrect**

Question: 5

class Car {  
String name = “Car”;  
String message() {  
return “from factory”;  
}  
}  
class Maruti extends Car {  
String name = “Alto800”;  
String message() {  
return “from showroom”;  
}  
}

public class Main {  
public static void main(String[] args) {  
Car c = new Car();  
System.out.println(c.name + ” ” + c.message());  
}  
}

* 1. 

a. Alto800 from showroom

* 1. Correct Answer



b. Car from showroom

* 1. Incorrect Answer



c. Car from factory

* 1. 

d. Altoo800 from factory

Answered

1. **Correct**

Question: 6

class Grandparent {  
public void print() {  
System.out.println(“Grandparent’s Print()”);  
}  
}  
class Parent extends Grandparent { }  
class Child extends Parent { }  
public class Main {  
public static void main(String[] args) {  
Child child = new Child();  
child.print();  
}  
}

* 1. Correct Answer



a.Grandparent’s Print()

* 1. 

b.CTE

* 1. 

c.no output

* 1. 

d. RTE

Answered

1. **Incorrect**

Question: 7

Q7. class Test  
{  
void nonStaticMethod()  
{  
System.out.println(“Non Static Method”);  
}  
}

public class Main  
{  
public static void main(String[] args)  
{  
Test test = null;  
test.nonStaticMethod();  
}  
}

* 1. 

a) Non Static Method

* 1. Correct Answer



b) RTE: NullPointerException

* 1. Incorrect Answer



c) Compile-time error

* 1. 

d) No output

Answered

1. **Incorrect**

Question: 8

class AClass  
{  
static void displayMessage()  
{  
System.out.println(“ParentClass Display”);  
}  
}

class BClass extends AClass  
{  
static void displayMessage()  
{  
System.out.println(“ChildClass Display”);  
}  
}

public class Main  
{  
public static void main(String[] args)  
{  
AClass aClass = null;  
aClass.displayMessage();  
}  
}

* 1. Correct Answer



a. ParentClass Display

* 1. 

b. ChildClass Display

* 1. 

c. Compile Time Error

* 1. Incorrect Answer



d. Run Time Error

Answered

[Continue](https://codehs.com/student/4767136/section/512694/assignment/131113602)

1. WAP for the below requirement:

  Create a class

  Create a NSV in the class.

  Create a sub class

  Create NSV in sub class

  Create NSM in sub class

  Print the super class NSV and Sub class NSV inside the method

  Call the method from a main class which contains main method

**class** Class1

{

**int** num1=25;

}

**public** **class** MyProgram **extends** Class1

{

**int** num=50;

**void** meth1()

{

System.***out***.println("Super class NSV :"+num1);//this.num1 //super.num1

System.***out***.println("Sub class NSV :"+num); //this.num

}

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

{

MyProgram mp=**new** MyProgram();

mp.meth1();

}

}

**OUTPUT:**

Super class NSV :25

Sub class NSV :50

WAP for the below requirement:

  Create 2 NSV in super class

  Create 1 NSV & 1 NSM in sub class

  Display the super class & sub class variable inside NSM

  Create object and call the method from the main method

**class** Super

{

**int** num=10,num1=20;

}

**public** **class** MyProgram **extends** Super

{

**int** num=30;

**void** meth()

{

System.***out***.println("Super class variable num :"+**super**.num);

//s.num //where s= object of class super

System.***out***.println("Super class variable num1 :"+num1);

System.***out***.println("Sub class variable num :"+num);//this.num //mp.num

}

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

{

MyProgram mp=**new** MyProgram();

mp.meth();

}

}

**OUTPUT:**

Super class variable num :10

Super class variable num1 :20

Sub class variable num :30

WAP for the below requirement:

  Create NSV in super class

  Create NSV in sub class

  Create NSM in next level of sub class.

  Print the variables inside the method

  Call the method from the main method

**class** Super

{

**int** nsv=15;

}

**class** Super1 **extends** Super

{

**int** nsv1=20;

}

**public** **class** MyProgram **extends** Super1

{

**void** meth()

{

System.***out***.println("Grand parent NSV :"+nsv);//new Super().nsv//this. super.

System.***out***.println("Parent NSV :"+nsv1);//super.nvs //this.nsv

}

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

{

MyProgram mp=**new** MyProgram();

mp.meth();

}

}

**OUTPUT:**

Grand parent NSV :15

Parent NSV :20

WAP for the below requirement: Create a NSM in super class Create 3 sub classes, each sub class should have NSM which will call superclass method. Create sub class objects in main method and call the class methods.

**class** Super

{

**void** methd()

{

System.***out***.println("Super class method :");

}

}

**class** Sub1 **extends** Super

{

**void** methd1()

{

methd();

}

}

**class** Sub2 **extends** Sub1

{

**void** methd2()

{

methd1();

}

}

**public** **class** MyProgram **extends** Sub2

{

**void** meth3()

{

methd2();

}

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

{

MyProgram mp=**new** MyProgram();

mp.meth3();

}

}

**OUTPUT:**

Super class method :

**JB\_OOP\_Inheritance\_MCQ\_Programs**

# MCQ

1. **Correct**

Question: 1

* 1. What are the features reused using Inheritance in Java?
  2. 

a) Methods

* 1. 

b) Variables

* 1. 

c) Constants

* 1. Correct Answer



d) All the above

Answered

1. **Correct**

Question: 2

The class that is being inherited or subclassed is called \_.

* 1. Correct Answer



a) Subclass

* 1. 

b) Superclass

Answered

1. **Correct**

Question: 3

You should use Inheritance when there is an IS-A relationship  
between classes. State TRUE or FALSE

* 1. Correct Answer



true

* 1. 

false

Answered

1. **Correct**

Question: 4

Find Superclass and Subclass in the below Java code snippet?  
class B  
{  
void show(){}  
}  
class A  
{  
void hide(){}  
}

* 1. 

a) B is superclass and A is subclass.

* 1. 

b) A is superclass and B is a subclass.

* 1. Correct Answer



c) There is no superclass or subclass present.

* 1. 

d) None

Answered

1. **Correct**

Question: 5

Find Superclass and Subclass in the below Java program?  
class Liquid  
{  
void pour(){}  
}  
class Juice extends Liquid  
{  
void filter(){}  
}

* 1. Correct Answer



a) The Liquid is a superclass and Juice is a subclass.

* 1. 

b) The Liquid is a Subclass and Juice is a Superclass.

* 1. 

c) There is no superclass or subclass

* 1. 

d) None

Answered

1. **Correct**

Question: 6

Which is the keyword used to implement inheritance in Java?

* 1. Correct Answer



a) extends

* 1. 

b) implements

* 1. 

c) instanceof

* 1. 

d) None

Answered

1. **Correct**

Question: 7

What will be the output of the following Java program?  
class A  
{  
int i;  
void display()  
{  
System.out.println(i);  
}  
}  
class B extends A  
{  
int j;  
void display()  
{  
System.out.println(j);  
}  
}  
class inheritance\_demo  
{  
public static void main(String args[])  
{  
B obj = new B();  
obj.i=1;  
obj.j=2;  
obj.display();  
}  
}

* 1. 

a) 0

* 1. 

b) 1

* 1. Correct Answer



c) 2

* 1. 

d) Compilation Error

Answered

1. **Correct**

Question: 8

What will be the output of the following Java program?

class A

{

int i;

}

class B extends A

{

int j;

void display()

{

super.i = j + 1;

System.out.println(j + " " + i);

}

}

class inheritance

{

public static void main(String args[])

{

B obj = new B();

obj.i=1;

obj.j=2;

obj.display();

}

}

* 1. 

a) 2 2

* 1. 

b) 3 3

* 1. Correct Answer



c) 2 3

* 1. 

d) 3 2

Answered

1. **Correct**

Question: 9

What will be the output?  
class Base  
{  
int x = 19;  
}

class Child extends Base  
{  
int x = 20;  
void shows()  
{  
System.out.println(“The base class data member (x) by Super Keyword :” + super.x);  
System.out.println(“The child class data member :” + x);

}

public static void main(String... a)

{

Child obj = new Child();

obj.shows();

}

}

* 1. Correct Answer



a) The base class data member (x) by Super Keyword :19  
The child class data member :20

* 1. 

b) The base class data member (x)by Super Keyword :0  
The child class data member :0

Answered

1. **Correct**

Question: 10

class X  
{  
public void A()  
{  
System.out.println(“123”);  
}  
}  
class Y extends X  
{  
public void B()  
{  
System.out.println(“456”);  
}  
}  
class Z extends X  
{  
public void C()  
{  
System.out.println(“789”);  
}  
}  
class V extends Y  
{  
public void D()  
{  
System.out.println(“10”);  
}  
}  
public class Hybrid1  
{  
public static void main(String[] args)  
{  
Y obj=new Y();  
V obj2=new V();  
obj.A();  
obj2.B();  
}  
}

* 1. 

a)  
123  
10

* 1. 

b)  
789  
10

* 1. 

c)  
456  
123

* 1. Correct Answer



d)  
123  
456

Answered

[Continue](https://codehs.com/student/4767136/section/512694/assignment/131244097)

Write a program by using a class name Calculator with the following specifications:

Class name: Calculator

Data Members/Instance variables: private int result

Member Methods:

Calculator() : constructor to initialize result with zero

void add(int a) to add a with the result

void sum(int a): to subtract a from the result

void mul() : to multiply a with the result

void div(int a): to divide result by a

void display() to display result

void clear() : to clear the result

**class** Calculator

{

**private** **int** result;

Calculator(**int** a)

{

result=a;

}

**public** **void** add(**int** a)

{

System.***out***.println("Adding a="+a+" and result="+result+" is :"+(a+result));

}

**void** sub(**int** a)

{

System.***out***.println("Subtracting a="+a+" from the result="+result+" is :"+(result-a));

}

**void** mul(**int** a)

{

System.***out***.println("to multiply a="+a+" with the result="+result+" is :"+a\*result);

}

**void** div(**int** a)

{

System.***out***.println("to divide result="+result+" by a="+a+" is :"+result/a);

}

**void** display()

{

System.***out***.println("to display result="+result);

}

**void** clear()

{

System.***out***.println("to clear the result");

}

}

**public** **class** MyProgram

{

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

{

Calculator c=**new** Calculator(50);

c.add(10);

c.sub(10);

c.mul(10);

c.div(10);

c.display();

c.clear();

}

}

**OUTPUT:**

Adding a=10 and result=50 is :60

Subtracting a=10 from the result=50 is :40

to multiply a=10 with the result=50 is :500

to divide result=50 by a=10 is :5

to display result=50

"A number is said to be Palindrome, if it appears to be the same

after reversing its digits". Write a program by using a class with

the following specifications:

Class name: Number

Data Members:

private int num;

Member Methods:

Number(int x)

: constructor to initialize num with x;

int reverse(int n): used to return the reverse of number.

void isPalindrome():to check and print whether the number is

          palindrome or not

**class** Number

{

**private** **int** num;

Number(**int** x)

{

num=x;

}

**int** reverse(**int** n)

{

**int** rev=0;

**while**(n!=0)

{

rev=rev\*10+(n%10);

n/=10;

}

**return** rev;

}

**void** isPalindrome()

{

**if**(num==reverse(num))

System.***out***.println("Number is palindrome ");

**else** System.***out***.println("Number is not a palindrome ");

}

}

**public** **class** MyProgram

{

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

{

Number n=**new** Number(101);

n.isPalindrome();

}

}

**OUTPUT:**

Number is palindrome

Vidyapati river bridge has a toll booth. Some cars passing through

the bridge stop to pay a toll of 5 and are termed as good cars but

some run through and do not pay any toll and are termed as bad cars,

"Write a class program with the following specifications:

Class name: Tollbooth

Data Members : private static int ncars, amount

Member Methods:

Tollbooth():constructor to initialize ncars and amount with 0.

void goodcars() : to increase only the ncars by 1

void badcars(): to increase ncars by 1 and amount by 5

void display() to print the number of cars that passed through

     the bridge within a defined period and the amount collected

**class** Tollbooth

{

**private** **static** **int** *ncars*,*amount*;

Tollbooth(**int** car)

{

*ncars*=car;

*amount* =0;

}

**void** goodcars()

{

*ncars*++;

*amount*+=5;

}

**void** badcars()

{

*ncars*++;

}

**void** display()

{

System.***out***.println("no of cars passed through thr bridge :"+*ncars*);

System.***err***.println("amount collected :"+*amount*);

}

}

**public** **class** MyProgram

{

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

{

Tollbooth tb=**new** Tollbooth(0);

**int** x=1;

**while**(x<=5)

{

tb.goodcars();

x++;

}

x=1;

**while**(x<=6)

{

tb.badcars();

x++;

}

tb.display();

}

}

**OUTPUT:**

no of cars passed through thr bridge :11

amount collected :25

**--------------------------------------------------------------------------------------------**

**JB\_OOP\_Abstract\_Interface\_and\_InnerClasses**

Animal - Mammal, Bird, and Reptile

Consider a zoo management system where you need to represent three types of animals:

Mammals, Birds, and Reptiles. Each type of animal has specific attributes and behaviors.

Implement an abstract class named Animal with an abstract method makeSound().

Create three subclasses: Mammal, Bird, and Reptile, each inheriting from the

Animal class.

Within the Mammal class, implement a static inner class named FoodPreference

that contains static methods to describe food preferences common to all mammals.

Include methods such as getPreferredFood() and setPreferredFood().

Within the Bird class, implement a non-static inner class named NestingHabit

that contains methods specific to bird nesting behaviors. Include methods such

as buildNest() and layEggs().

Within the Reptile class, implement multiple methods such as baskInSun(),

hibernate(), and layEggs() directly within the class.

Override the makeSound() method in each subclass to provide a unique sound for

each type of animal.

Create instances of each animal type (e.g., lion, eagle, snake) and demonstrate the

functionalities implemented within their respective classes.

Method Information:

Animal class:

Method: makeSound()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: makeSound

Mammal class:

Method: makeSound()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: makeSound

Inner Static Class: FoodPreference

Method: getPreferredFood()

Return Type: String

Access Modifier: public, static

Parameters: None

Method Name: getPreferredFood

Method: setPreferredFood()

Return Type: void

Access Modifier: public, static

Parameters: String food

Method Name: setPreferredFood

Bird class:

Method: makeSound()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: makeSound

Inner Non-Static Class: NestingHabit

Method: buildNest()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: buildNest

Method: layEggs()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: layEggs

Reptile class:

Method: makeSound()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: makeSound

Method: baskInSun()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: baskInSun

Method: hibernate()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: hibernate

Method: layEggs()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: layEggs

**abstract** **class** Animal

{

**public** **abstract** **void** makeSound();

}

**class** Mammal **extends** Animal//NOC

{

@Override

**public** **void** makeSound()

{

System.***out***.println("Make sound :Roare");

}

**static** **class** FoodPreference//SIC

{

**static** String *foods*;

**public** **static** String getPreferredFood()

{

**return** *foods*;

}

**public** **static** **void** setPreferredFood(String food)

{

*foods*=food;

}

}

}

**class** Bird **extends** Animal

{

@Override

**public** **void** makeSound()

{

// **TODO** Auto-generated method stub

System.***out***.println("Make sound : whistling");

}

**class** NestingHabit

{

**public** **void** buildNest()

{

System.***out***.println("Build Nest : using wood sticks");

}

**public** **void** layEggs()

{

System.***out***.println("Lays eggs in Nest");

}

}

}

**class** Reptile **extends** Animal

{

@Override

**public** **void** makeSound()

{

// **TODO** Auto-generated method stub

System.***out***.println("Make sound : silent and hissing");

}

**public** **void** baskInSun()

{

System.***out***.println("Bask In Sun :returning home at evening to their selter");

}

**public** **void** hibernate()

{

System.***out***.println("Hiberante : In winter");

}

**public** **void** layEggs()

{

System.***out***.println("lays eggs :Underground");

}

}

**public** **class** MyProgram// Main

{

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

{

Animal lion =**new** Mammal();

Animal eagle=**new** Bird();

Animal snake=**new** Reptile();

System.***out***.println("---------------Animal-------Parent\_class--------------------");

System.***out***.println("---------------Mammal-------Lion----------------------------");

lion.makeSound();

Mammal.FoodPreference mf=**new** Mammal.FoodPreference();

mf.*setPreferredFood*("Flesh");

System.***out***.println(mf.*getPreferredFood*());

System.***out***.println("---------------Bird-------Eagle----------------------------");

eagle.makeSound();

Bird.NestingHabit bn=**new** Bird().**new** NestingHabit();

bn.buildNest();

bn.layEggs();

System.***out***.println("---------------Reptile-------Snake----------------------------");

snake.makeSound();

Reptile r=**new** Reptile();

r.baskInSun();

r.hibernate();

r.layEggs();

}

}

Library Management System

You are tasked with designing a library management system to handle different types

of items, including books and DVDs. Your system should support functionalities such

as borrowing, returning, and calculating late fees for overdue items.

Implement an interface named Borrowable with methods borrow() and returnItem().

Create an abstract class named LibraryItem that implements the Borrowable interface

and includes common attributes like title, available status, borrowed status, and

daysOverdue representing the number of days the item is overdue.

Develop two concrete subclasses of LibraryItem: Book and DVD, each inheriting

from the LibraryItem abstract class.

Within the Book class, create a static inner class named BookGenre with a method

setGenre() that allows setting the genre of the book.

For the DVD class, implement a non-static inner class named DVDType with a method

setType() to set the type of the DVD.

Override the borrow() and returnItem() methods in each subclass to handle borrowing

and returning of respective items, updating the available status, borrowed status,

and daysOverdue accordingly.

Implement the calculateLateFee() method in each subclass to calculate the late fee

for overdue items. For books, the late fee is 50Rs. per day overdue, and for DVDs,

it's 10Rs per day overdue.

Implement the printDetails() method in each subclass to print details of the item

including title, author/director, and availability status.

Demonstrate the functionalities by creating instances of Book and DVD objects,

borrowing and returning items, and calculating late fees for overdue items.

Method Information:

Borrowable interface:

Method: borrow()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: borrow

Method: returnItem()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: returnItem

LibraryItem abstract class:

Method: borrow()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: borrow

Method: returnItem()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: returnItem

Method: calculateLateFee()

Return Type: double

Access Modifier: abstract

Parameters: None

Method Name: calculateLateFee

Method: printDetails()

Return Type: void

Access Modifier: abstract

Parameters: None

Method Name: printDetails

Book class:

Method: calculateLateFee()

Return Type: double

Access Modifier: public

Parameters: None

Method Name: calculateLateFee

Method: printDetails()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: printDetails

Inner Static Class: BookGenre

Method: setGenre()

Return Type: BookGenre (for method chaining)

Access Modifier: public, static

Parameters: String genre

Method Name: setGenre

DVD class:

Method: calculateLateFee()

Return Type: double

Access Modifier: public

Parameters: None

Method Name: calculateLateFee

Method: printDetails()

Return Type: void

Access Modifier: public

Parameters: None

Method Name: printDetails

Inner Non-Static Class: DVDType

Method: setType()

Return Type: DVDType (for method chaining)

Access Modifier: public

Parameters: String type

Method Name: setType

**import** java.util.Scanner;

//----------------------------interface Borrowable--------------------------------------

**interface** Borrowable //PC

{

**public** **void** borrow();

**public** **void** returnItem();

}

//--------------------------class LibraryItem------------------------------------------

**abstract** **class** LibraryItem **implements** Borrowable //SC & PC

{

String title;

**boolean** available;

//status,

**boolean** borrowed ;

//status, and

**int** daysOverdue;

**public** LibraryItem(String title)

{

**this**.title = title;

**this**.available = **true**;

**this**.borrowed = **false**;

**this**.daysOverdue = 0;

}

@Override

**public** **void** borrow()

{

// **TODO** Auto-generated method stub

**if**(available)

{

available=**false**;

borrowed = **true**;

System.***out***.println(title + " has been borrowed.");

}

**else**

{

System.***out***.println(title + " is not available for borrowing.");

}

}

@Override

**public** **void** returnItem()

{

// **TODO** Auto-generated method stub

**if** (borrowed)

{

available = **true**;

borrowed = **false**;

System.***out***.println(title + " has been returned.");

} **else**

{

System.***out***.println(title + " was not borrowed.");

}

}

**abstract** **double** calculateLateFee();

**abstract** **void** printDetails();

//title, available status, borrowed status, and daysOverdue

}

//------------------------Book ---------------------------------------------------

**class** Book **extends** LibraryItem //SC

{

String author;

**public** Book(String title, String author) //

{

**super**(title);

**this**.author = author;

}

**static** **class** BookGenre//SIC

{

**public** **static** BookGenre setGenre(String genre) //(for method chaining)

{

//allows setting the genre of the book

**return** **new** BookGenre();

}

}

@Override

**public** **double** calculateLateFee()

{

**int** od;

// the late fee is 50Rs.per day overdue

System.***out***.print("Enter no of Days overdue After Borrowing book:-");

Scanner sc=**new** Scanner(System.***in***);

od=sc.nextInt();

daysOverdue=od;

**return** 50.0\*od;

}

@Override

**public** **void** printDetails()//

{

//print details of the item including title, author/director, and availability status.

System.***out***.println("Book Details:-");

System.***out***.println("Title :-"+title);

System.***out***.println("Author :-"+author);

System.***out***.println("availability status :-"+available);

System.***out***.println("borrowed status :-"+borrowed);

System.***out***.println("daysOverdue :-"+daysOverdue);

}

@Override

**public** **void** borrow() //

{

// **TODO** Auto-generated method stub

**if**(available)

{

available=**false**;

borrowed = **true**;

System.***out***.println(title + " has been borrowed.");

}

**else**

{

System.***out***.println(title + " is not available for borrowing.");

}

}

@Override

**public** **void** returnItem()

{

// **TODO** Auto-generated method stub

**if** (borrowed)

{

available = **true**;

borrowed = **false**;

System.***out***.println(title + " has been returned.");

} **else**

{

System.***out***.println(title + " was not borrowed.");

}

}

}

//-----------------------DVD----------------------------------------

**class** DVD **extends** LibraryItem//SC

{

String author;

**public** DVD(String title, String author)

{

**super**(title);

**this**.author = author;

}

@Override

**public** **double** calculateLateFee()

{

**int** od;

// the late fee is 50Rs.per day overdue

System.***out***.print("Enter no of Days overdue After Borrowing DVD:-");

Scanner sc=**new** Scanner(System.***in***);

od=sc.nextInt();

daysOverdue=od;

**return** 10.0\*od;

//10Rs per day overdue

}

@Override

**public** **void** printDetails()

{

//print details of the item including title, author/director, and availability status.

System.***out***.println("DVD Details:-");

System.***out***.println("Title :-"+title);

System.***out***.println("Author :-"+author);

System.***out***.println("availability status :-"+available);

System.***out***.println("borrowed status :-"+borrowed);

System.***out***.println("daysOverdue :-"+daysOverdue);

}

**class** DVDType//IN-SC

{

**public** DVDType setType(String type)//(for method chaining)

{

//set the type of the DVD

**return** **new** DVDType();

}

}

@Override

**public** **void** borrow()

{

// **TODO** Auto-generated method stub

**if**(available)

{

available=**false**;

borrowed = **true**;

System.***out***.println(title + " has been borrowed.");

}

**else**

{

System.***out***.println(title + " is not available for borrowing.");

}

}

@Override

**public** **void** returnItem()

{

// **TODO** Auto-generated method stub

**if** (borrowed)

{

available = **true**;

borrowed = **false**;

System.***out***.println(title + " has been returned.");

} **else**

{

System.***out***.println(title + " was not borrowed.");

}

}

}

//---------------------Main class-----------------------------------

**public** **class** MyProgram// Main

{

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

{

System.***out***.println("----------------------LibraryItem-------------------------------------");

System.***out***.println("---------------------Book---------------------------------------------");

Book b=**new** Book("java edition 21C","jems goslin");

System.***out***.println("---------------------------------->Before borrowing");

b.printDetails();

System.***out***.println("---------------------------------->After borrowing");

b.borrow();

b.printDetails();

System.***out***.println("----------------------------------->Returning book");

b.returnItem();

System.***out***.println("------------------------------------>Late fee");

System.***out***.println("the late fee is 50Rs.per day :-"+ b.calculateLateFee());

System.***out***.println("------------------------------------>after returning");

b.printDetails();

System.***out***.println("---------------------DVD---------------------------------------------");

DVD d=**new** DVD("Spidrman","peter parkar");

System.***out***.println("---------------------------------->Before borrowing");

d.printDetails();

System.***out***.println("---------------------------------->After borrowing");

d.borrow();

d.printDetails();

System.***out***.println("----------------------------------->Returning book");

d.returnItem();

System.***out***.println("------------------------------------>Late fee");

System.***out***.println("the late fee is 10Rs.per day :-"+ d.calculateLateFee());

System.***out***.println("------------------------------------>after returning");

d.printDetails();

}

}

**OUTPUT:**

----------------------LibraryItem-------------------------------------

---------------------Book---------------------------------------------

---------------------------------->Before borrowing

Book Details:-

Title :-java edition 21C

Author :-jems goslin

availability status :-true

borrowed status :-false

daysOverdue :-0

---------------------------------->After borrowing

java edition 21C has been borrowed.

Book Details:-

Title :-java edition 21C

Author :-jems goslin

availability status :-false

borrowed status :-true

daysOverdue :-0

----------------------------------->Returning book

java edition 21C has been returned.

------------------------------------>Late fee

Enter no of Days overdue After Borrowing book:-5

the late fee is 50Rs.per day :-250.0

------------------------------------>after returning

Book Details:-

Title :-java edition 21C

Author :-jems goslin

availability status :-true

borrowed status :-false

daysOverdue :-5

---------------------DVD---------------------------------------------

---------------------------------->Before borrowing

DVD Details:-

Title :-Spidrman

Author :-peter parkar

availability status :-true

borrowed status :-false

daysOverdue :-0

---------------------------------->After borrowing

Spidrman has been borrowed.

DVD Details:-

Title :-Spidrman

Author :-peter parkar

availability status :-false

borrowed status :-true

daysOverdue :-0

----------------------------------->Returning book

Spidrman has been returned.

------------------------------------>Late fee

Enter no of Days overdue After Borrowing DVD:-2

the late fee is 10Rs.per day :-20.0

------------------------------------>after returning

DVD Details:-

Title :-Spidrman

Author :-peter parkar

availability status :-true

borrowed status :-false

daysOverdue :-2

Suppose you are tasked with designing a banking system. Implement the following

classes/interfaces:

BankAccount (abstract class):

Attributes: accountNumber, balance

Methods:

deposit(double amount) - adds the given amount to the balance.

withdraw(double amount) - deducts the given amount from the balance.

abstract void monthlyUpdate() - updates the balance at the end of each month.

Different types of accounts might have different rules for this update.

toString() - overrides the toString() method to display the account details.

--------------------------------------------------------------------------------------

SavingsAccount (subclass of BankAccount):

Additional Attributes: interestRate

Additional Methods:

monthlyUpdate() - updates the balance by adding the interest accrued.

--------------------------------------------------------------------------------------

CurrentAccount (subclass of BankAccount):

Additional Attributes: overdraftLimit

Additional Methods:

monthlyUpdate() - updates the balance by charging an overdraft fee if necessary.

------------------------------------------------------------------------------------

Customer (interface):

Methods:

void depositToAccount(BankAccount account, double amount) - allows a customer to

          deposit money into their account.

void withdrawFromAccount(BankAccount account, double amount) - allows a customer to

         withdraw money from their account.

-----------------------------------------------------------------------------------------

Create one more clas Bank in same program and create :

Inner Class: Transaction

Attributes: transactionId, transactionDate, amount, type (e.g., deposit, withdrawal),

                     accountNumber

Initializes all attributes using static method or constructor.

Methods:

toString() - overrides the toString() method to display the transaction details.

Methods:

void processTransaction(BankAccount account, double amount, String type) - processes a

transaction for a given account (deposit/withdrawal).

void displayTransactions() - displays all transactions performed in the bank.

Ensure appropriate constructors are implemented for each class. Also, override the

toString() method in each class to display relevant information when printed.

Your task is to implement the above classes/interfaces with appropriate methods

and attributes, demonstrating the use of inheritance, interfaces, inner classes,

constructor invocation, and method overriding. Additionally, provide a simple

demonstration of how these classes/interfaces can be used together in a banking

system.