IRC_SKCT_Java2_CY_COD_Polymorphism

Test Summary

- No. of Sections: 1
- No. of Questions: 5
- Total Duration: 120 min

Section 1 - Coding

Section Summary

- No. of Questions: 5
- Duration: 120 min

Additional Instructions:

None

Q1. **OVERLOADING THE MAIN FUNCTION**

1. Overload the main method by passing a single String.
2. Overload the main method by passing a two String.

Input Format

No console input.

Output Format

The first line of the output should display 'Hi'.
The second line of the output should display 'Overloaded: Hello World'.
The third line of the output should display 'Overloaded: Tom & Jerry'.

Sample Input

Sample Output

```
Hi
Overloaded: Hello World
Overloaded: Tom & Jerry
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q2. **METHOD OVERLOADING USING TYPE CONVERSION**

Create a class named **'Main'**. Define a method **'print'**

1. Create an object obj.
2. Call method 'print' with one argument in an Integer type, Output should display given Integer.
3. Call method 'print' with one argument in a String type, Output should display given String.
4. Call method 'print' with one argument in a Boolean type, Output should display given Boolean.

Input Format

No console input.

Output Format

The first line of the output should display 35
The second line of the output should display 'Hello World'
The third line of the output should display 24.35

Sample Input

Sample Output

```
35
Hello World
24.35
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q3. Write a program to illustrate dynamic polymorphism, create two classes Vehicle and Motorbike. Motorbike inherits the Vehicle class.

Create a method move() in base class that takes a string as input and print them.
Override the method move() in derived class that also takes a string as input and prints them.

Input Format

Input two strings in separate line

Output Format

Displays the string after execution.
Refer sample output

Constraints

Only strings.

Sample Input

```
move fast
vehicles
```

Sample Output

```
vehicles
move fast
```

Sample Input

```
are sweet
mangoes
```

Sample Output

```
mangoes
are sweet
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q4. Create a parent class that consists of two methods m1 and m2.
m1 doesn't take any arguments and it just prints from parent.
m2 takes an integer value as parameter and prints the same.
Create a child class that extends parent class and override its methods.
m1 doesn't take any arguments and it just prints from child.
m2 takes an integer value as parameter and prints the same.
In the main class, create objects for the above classes and call the corresponding methods.

Input Format

The input consists of the integer value for both the classes separated by a space.

Output Format

The output displays the result. Refer sample output.

Constraints

integers only.

Sample Input

```
1 2
```

Sample Output

```
From parent m1()
1
From child m1()
2
```

Sample Input

```
2 4
```

Sample Output

```
From parent m1()
2
From child m1()
4
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Q5. **Function Overloading**
An ice-cream vendor sells his ice-creams in cone(radius r and height h) and ball(radius r) shaped containers. However, he is unsure of the quantity that can be filled in the two containers. You are required to write a program in java that prints the volume of the containers given its respective dimensions as input. Your class must be named 'Icecream' which has two methods with same name 'Quantity' each having the respective dimensions of the containers as the parameters.
Function Header:
public void Quantity(int r, int h)
public void Quantity(int r)

Input Format

If the quantity of the cone is to be calculated, the input must have the radius(r) and height(h) in the same line separated by a space.

For calculating the quantity of the ball, the input must have its radius(r).

Note: Input type should be integer.

Output Format

The output must display the volume of the container rounded off to two decimal places for which the dimensions are given.

Sample Input

```
4 5
```

Sample Output

```
82.90
```

Sample Input

```
4
```

Sample Output

```
267.28
```

Time Limit: - ms Memory Limit: - kb Code Size: - kb

Answer Key & Solution

Section 1 - Coding

Q1

Test Case

Input	Output
	Hi Overloaded: Hello World Overloaded: Tom & Jerry

Weightage - 100

Sample Input

Sample Input	Sample Output
	Hi Overloaded: Hello World Overloaded: Tom & Jerry

Solution

Header

```
class Main{



public static void main(String s){
    System.out.println("Overloaded: "+s);
}

public static void main(String s1,String s2){
    System.out.println("Overloaded: "+s1+" & "+s2);
}
```

Footer

```
public static void main(String args[]){
    System.out.println("Hi");
    Main.main("Hello World");
    Main.main("Tom","Jerry");
}
}
```

Q2

Test Case

Input	Output
	35 Hello World 24.35

Weightage - 100

Sample Input

Sample Input	Sample Output
	35 Hello World

	24.35

Solution

Header

```
class Main{



public int print(int a){
    return a;
}
public String print(String a){
    return a;
}
public Double print(Double a){
    return a;
}
```

Footer

```
public static void main(String args[]){
    Main m=new Main();
    System.out.println(m.print(35));
    System.out.println(m.print("Hello World"));
    System.out.println(m.print(24.35));
}
}
```

Q3

Test Case

Input

```
fire
rapid
```

Output

```
rapid
fire
```

Weightage - 10

Input

```
fever
yellow
```

Output

```
yellow
fever
```

Weightage - 10

Input

```
rose
red
```

Output

```
red
rose
```

Weightage - 10

Input

Output

```
comb
honey
```

```
honey
comb
```

Weightage - 10

Input

Output

```
the limit
do not cross
```

```
do not cross
the limit
```

Weightage - 10

Input

Output

```
goes on
life
```

```
life
goes on
```

Weightage - 10

Input

Output

```
berries
blue
```

```
blue
berries
```

Weightage - 20

Input

Output

```
mare
night
```

```
night
mare
```

Weightage - 20

Sample Input

Sample Output

```
move fast
vehicles
```

```
vehicles
move fast
```

Sample Input

Sample Output

```
are sweet
mangoes
```

```
mangoes
are sweet
```

Solution

Header

```
import java.util.Scanner;


class Vehicle{
```

```
    public void move(String t){
    System.out.println(""+t);
    }
}
class MotorBike extends Vehicle{
    public void move(String t1){
    System.out.println(""+t1);
    }
}
```

Footer

```
class Poly{

    public static void main(String[] args){

    Vehicle vh=new MotorBike();
    Scanner sc=new Scanner(System.in);
    String t,t1;
    t=sc.nextLine();
    t1=sc.nextLine();
    vh.move(t1);
    vh=new Vehicle();

    vh.move(t);

    }

}
```

Q4

Test Case

Input	Output
5 7	From parent m1() 5 From child m1() 7

Weightage - 10

Input	Output
10 20	From parent m1() 10 From child m1() 20

Weightage - 10

Input	Output
22 33	From parent m1() 22

```
From child m1()
```

Weightage - 10

Input

```
58 67
```

Output

```
From parent m1()
58
From child m1()
67
```

Weightage - 10

Input

```
65 80
```

Output

```
From parent m1()
65
From child m1()
80
```

Weightage - 10

Input

```
9 3
```

Output

```
From parent m1()
9
From child m1()
3
```

Weightage - 10

Input

```
7 6
```

Output

```
From parent m1()
7
From child m1()
6
```

Weightage - 20

Input

```
01 02
```

Output

```
From parent m1()
1
From child m1()
2
```

Weightage - 20

Sample Input

```
1 2
```

Sample Output

```
From parent m1()
1
From child m1()
2
```

Sample Input

```
2 4
```

Sample Output

```
From parent m1()
2
From child m1()
4
```

Solution

Header

```
import java.util.Scanner;



class Parent{
    protected void m1() { System.out.println("From parent m1()");}

    protected void m2(int a) { System.out.println(""+a); }
}

class Child extends Parent
{
    public void m1() { System.out.println("From child m1()");}
    public void m2(int b) { System.out.println(""+b);}

}
```

Footer

```
class Main
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        int a,b;
        a=sc.nextInt();
        b=sc.nextInt();
        Parent obj1 = new Parent();
        obj1.m1();
        obj1.m2(a);
        Parent obj2 = new Child();
        obj2.m1();
        obj2.m2(b);
    }
}
```

Q5

Test Case

Input

```
20
```

Output

```
33409.60
```

Weightage - 20

Input

```
3  2.5
```

Output

```
112.76
```

Weightage - 30

Input

```
2  6
```

Output

```
24.87
```

Weightage - 20

Input

```
6
```

Output

```
902.06
```

Weightage - 20

Input

```
2  0
```

Output

```
0.00
```

Weightage - 10

Sample Input

```
4  5
```

Sample Output

```
82.90
```

Sample Input

```
4
```

Sample Output

```
267.28
```

Solution

Header

```
import java.util.Scanner;
import java.math.*;
import java.text.DecimalFormat;
```

```
class Icecream{
    public double qty,qty_rd, pi = 3.14;
    DecimalFormat d = new DecimalFormat("0.00");
    public void Quantity(int r){
        qty = 1.33*pi*r*r*r;
        qty_rd = Math.round(qty * 100.0) / 100.0;
        System.out.println(d.format(qty_rd));
    }
    public void Quantity(int r, int h){
        qty = 0.33*pi*r*r*h;
        qty_rd = Math.round(qty * 100.0) / 100.0;
        System.out.println(d.format(qty_rd));
    }
}
```

Footer

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

```
        int r, h;
        Icecream ic = new Icecream();
        Scanner in = new Scanner(System.in);
        r = in.nextInt();
        if(in.hasNextInt())
        {
        h = in.nextInt();
        ic.Quantity(r,h);
        }
       else
        ic.Quantity(r);
    }
}
```