Muhammad Shoab Islam Rohan

Midterm Examination (Spring 2024)
Object Oriented Concepts

1. a) Explain Compile-time and run-time polymorphism with an example of each.

=> Polymorphism means "many forms". It allows one interface to be used for different types of objects.

Types of Polymorphism:

1. Compile-time polymorphism (Method Overloading)
2. Run-time polymorphism (Method Overriding)

Compile-time polymorphism happes when multiple methods have the same name but different parameters.

Example: Method Overloading in Java

Method overloading

```
class MathOperations {
    void add(int a, int b){
System.out.println("Sum (int): " +(a+b));
}

    void add(double a, double b){
System.out.println("Sum (double):"+(a+b));
}
```

Method with 2 int parameters

Method with 2 double parameters

```
    void add(int a, int b, int c){
    System.out.println("Sum (3 int): "+(a+b+c));
    }
}

public class CompileTimePoly {          [এই class টি পুরো program টি run করবে]
    public static void main(String[] args){ [যেহেতু method, Java program এর entry point]
    Calculator obj = new Calculator();
        obj.add(10, 5);          [add(int, int) কল হবে]
        obj.add(5.5, 2.2);       [add(double, double কল হবে]
        obj.add(1,2,3);          [add(int, int, int) কল হবে]
    }
}
```

Output:
Sum (int): 15
Sum (double): 7.7
Sum (3 int): 6

কেন compile-time polymorphism?
→ কোন ফাংশন রান হবে, তা compile time এই নির্ধারণ হয়।
→ Method name একই, কিন্তু parameter আলাদা।

Run-time polymorphism happens when a subclass provides a specific implementation of a method already defined in the parent class.

Example : Method Overriding in Java

Method Overriding

```
class Animal{
    void sound(){
     System.out.println("Animals make sounds");
    }
}
class Dog extends Animal{
    @Override
   void sound(){
     System.out.println("Dog barks");
    }
}
public class RunTimePoly {
    public static void main(String[] args){
     Animal myAnimal = new Dog();[// Parent class reference, Child class object]
     myAnimal.sound();      [//calls Dog's overridden method]
    }
}
```

Output: Dog barks

কেন Run-time Polymorphism?

=> কোন method run হবে, তা Run-time এ নির্ধারণ হয়।

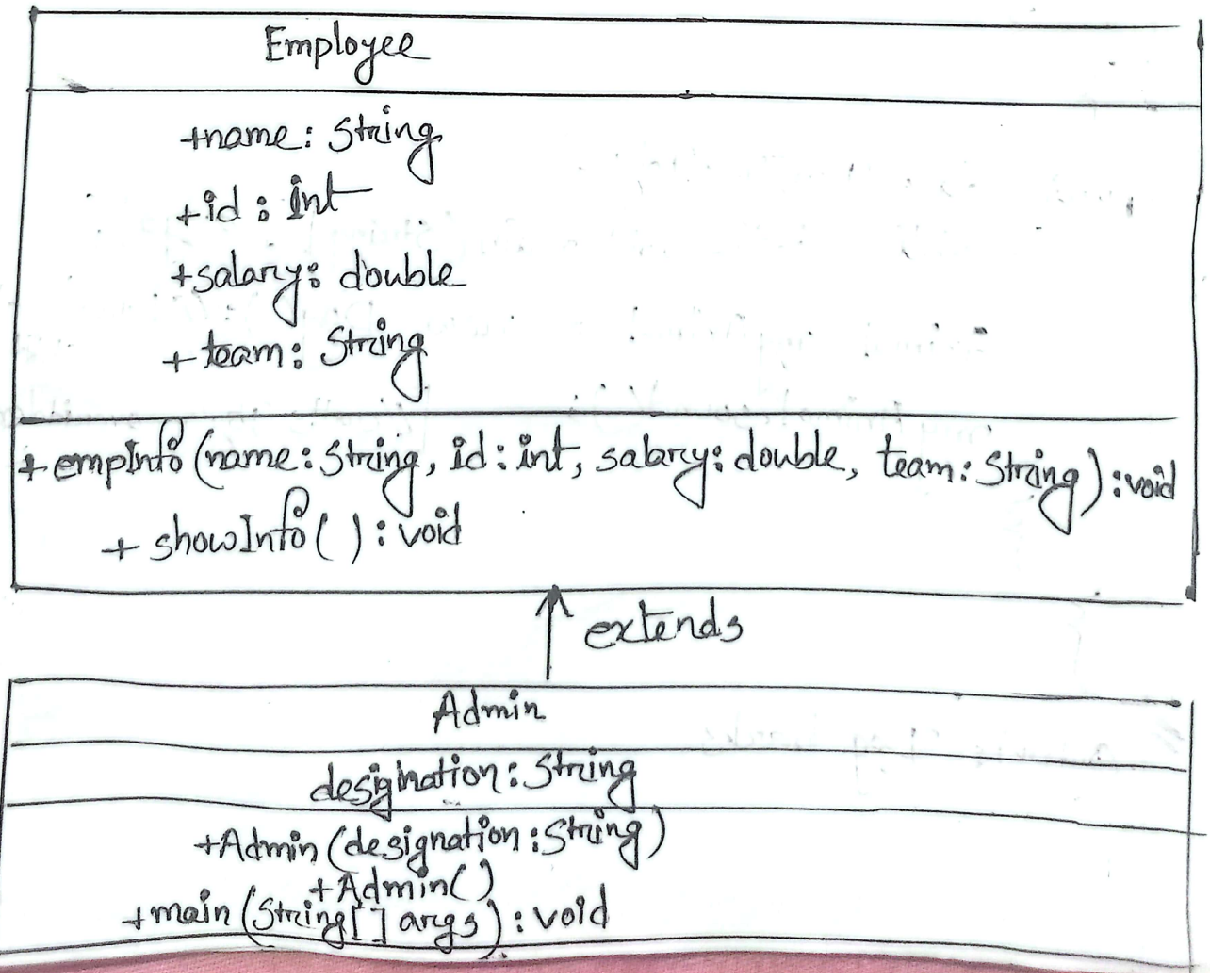
=> Method name ও parameters একই, কিন্তু child class নতুন করে method লিখেছে।

সংক্ষেপে:

Method Overloading:→ Compile-time (একই নাম, কিন্তু ভিন্ন প্যারামিটার)

Method Overriding:→ Run-time (একই নাম ও প্যারামিটার, কিন্তু child class নতুন করে define করে)

b) Convert the below UML Diagram into a Java Code. Create two objects and then implement all the methods and display the outputs.



Answer:

```
// Employee class (Base class)

class Employee{

        String name;
        int id;
      double salary;
      String team;
// Method to set employee Information
void emInfo (String name, int id, double salary, String team){
        this.name = name;
        this. id = id;
        this. salary = salary;
        this. team = team;

}

// Method to display employee information
void showInfo ( ){
    System.out. println ("Name: " +name);
    System.out. println ("ID: " +id);
    System. out. println ("Salary: " +salary);
    System. out. println ("Team: " + team);

  }
}
```

```
//Admin class (Derived class)
class Admin extends Employee{
    String designation;
//Constructor with designation parameter
  Admin(String designation){
     this.designation = designation;
  }
//Default constructor
  Admin(){
  this.designation = "Unknown";
  }
//Overriding showInfo method to include designation
@override
void showInfo(){
   super.showInfo(); //Call parent class method
System.out.println("Designation: " + designation);
}
```

```
public static void main (String [] args) {
    // Creating first Admin object
    Admin admin1 = new Admin ("Manager");
    admin1.empInfo ("Alice", 101, 50000, "HR");
    admin1.showInfo ();
    System.out.println ();

    // Creating second Admin object
    Admin admin2 = new Admin ("Supervisor");
    admin2.empInfo ("Bob", 102, 45000, "Finance");
    admin2.showInfo ();
}
}
```

2.a) Code Explanation:

1. Constructor Overloading:
   => Default constructor Car() :- Brand & model set as "Unknown"

=> Parameterized constructor Car (String brand, String model) :- Specific brand & model set করতে পারবে।

2. Method Overloading:

=> accelerate (int speedIncrease) : Speed বাড়ায়।

=> accelerate (int speedIncrease, int time) : Speed বাড়ায় with extra time calculation.

=> brake (int speedDecrease) : Speed কমায়, যদি 0 এর নিচে যায় তাহলে stop করে।

Full Java code:

⇒

```
class Car{
    private String brand;
    private String model;
    private int speed;

public Car(){                          → Default constructor
    this.brand = "Unknown";
    this.model = "Unknown";
    this.speed = 0;
}

public Car(String brand, String model){   → Parameterized Constructor
    this.brand = brand;
    this.model = model;
    this.speed = 0;
}

public void displayInfo(){
  System.out.println("Car: " + brand + " " + model);
}
```

Output show করবে

Accelerate Method (overloaded)

```
public void accelerate (int speedIncrease){
    speed += speedIncrease;
System.out.println("Accelerating... Speed is now: " + speed + "mph");
}
```

AM with Time Parameter (overloaded)

```
public void accelerate(int speedIncrease, Int time){
 speed += (speedIncrease + time);
System.out.println("Accelerating for" + time + "seconds... Speed
  is now:" + speed + "mph");
}
```

Brake Method

```
public void brake(int speedDecrease){
  speed -= speedDecrease;
  if(speed <= 0){
     speed = 0;
System.out.println("Braking... Car has stopped.");
} else{
 System.out.println("Breaking... Speed is now: " + speed +
 " mph");
  }
}
}
```

40 - 1(b), 2(d)

b) Types of Variables in Java

=> Java তে variables 3 type এর হয়,

1. Instance Variable (Object - specific)

=> প্রতিটি object আলাদা থাকে, Object create না করলে use করা যায় না, Class এর মধ্যে declare করা হয় কিন্তু Static keyword থাকে না। Heap memory তে store হয়।

Example:

```
class Car {
    String brand; [Instance variable]
    int speed;
```

প্রতিটি object এর instance variable এর নিজস্ব copy থাকে।

2. Static Variable (Class specific)

=> Class এর মধ্যে declare করা হয়, কিন্তু "static" keyword থাকে, Class এর সব object এর জন্য same value থাকে।

Object ছাড়াও access করা যায় (ClassName.variableName)

Example:

```
class Student {
    String name; [Instance variable]
    static String university = "DIU"; [Static variable]
```

3. Local Variable (Method specific)

=> Method এর ভিতর declare করা হয়, method এর বাইরে use করা যায় না

Example:

```
class Example{
    void show(){
        int x = 10;
    System.out.println("Value of x: " +x);
    }
}
```

c) Animal pet1 = new Dog();
Identify the type of the object 'pet1' in this above statement?

=> Animal pet1 = new Dog();

pet1 is declared as an object of type Animal, but এটা একটা Dog object কে refer করতেছে।

Explanation:

1. pet1 এর reference type টা হলো Animal (এই type দিয়ে pet1 কে declare করা হইছে)।

2. new Dog() দিয়ে একটা Dog object create করা হয়েছে, আর pet1 সেই object কে refer করতেছে।

এইক্ষেত্রে, Dog class Animal class এর subclass, তাই Dog একটা Animal type এর object হিসেবে treat করা যাবে।

Main Point:

Reference Type: Animal (pet1 এর type)

Actual Type: Dog (যে object create করা হয়েছে)

Polymorphism এর কারনে, যদি Dog class Animal class এর কোনো method কে override করে থাকে তাহলে pet1 দিয়ে Dog এর method call হবে (Upcasting এর জন্য)

Polymorphism

=> Polymorphism মানে একই নামের method different ways এ কাজ করতে পারে।

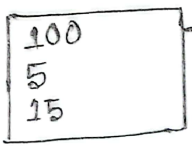
Types:

1. Method Overloading: Same method name, different parameters (Compile-time Polymorphism)
2. Method Overriding: Same method, different behavior in child class (Runtime Polymorphism).

d) Explanation of Execution:

1. solve1.puzzle1();

* First call to puzzle1() of JigSaw class:
  rubics = 100 is a local variable and printed: 100.
  instance = 5 (from JigSaw class), so it prints: 5.
  check = 15 (after modification in puzzle1()), so it prints: 15.



→ Output 1

2. solve2.puzzle2();

*Second call to puzzle2() from chess class:

super.puzzle2() is called first, which calls puzzle2() from Sudoku:

rubics = 200 (local variable in puzzle2() of Sudoku), prints: 200

instance = 9 (modified in puzzle2() of Sudoku), prints: 9.

check = 15 (modified in puzzle1()), prints: 15.

instance += 2 (increments instance), so now instance = 11.

3. After the call to super.puzzle2(), we are in chess:

rubics = 300 (local variable in puzzle2() of chess), prints: 300.

instance = 11 (modified by Sudoku), prints: 11.

check = 15 (static variable, same as before), prints: 15.

instance += 2 (increments instance), so now instance = 13.

যেহেতু solve2.puzzle2() finish হওয়ার পরে আমরা solve1.instance এর value print করেছি, তাই instance variable টা instance ই থাকবে, puzzle2() এর মধ্যে solve1 modify করা হয়নি, তাই remains value same থাকবে যা puzzle1() call করার পরে ছিল, which is still 5.

100
5
15
200
9
15
300
11
15

Instance value after puzzle2 : 5

Fall 2023

1 a) Already solved

b) Explanation:-

1. double d1 = 100.04;

double variable d1 is initialized with the value 100.04.

2. int i1 = 5;

int variable i1 is initialized with the value 5.

3. int i2 = (int)d1;

The double value d1 (100.04) is explicity cast to int. This casting removes the decimal part and only stores the integer part of d1, which is 100. Therefore, i2 will hold the value 100.

4. double d2 = (double) i1;

The int value i1 (5) is explicity cast to double. This doesn't change the value; it just converts i1 to 5.0 in double precision.

Therefore, d2 will hold the value 5.0.

5. System.out.println ("Double value" + d2);

This prints the value of d2 which is 5.0, so the output will be:

Double value 5.0

6. System.out.println ("Integer value" + i2);

This prints the value of i2 which is 100, so the output will be:

Integer value 100.

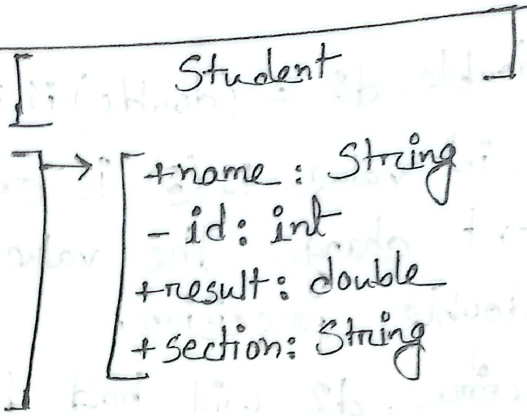
Final Output:

Double value 5.0.

Integer value 100.

c) public class Student {

```
    public String name;
    private int id;
    public double result;
    public String section;
```




```
public void storeInfo(String name, int id, double result,
                      String section) {
    this.name = name;
    this.id = id;
    this.result = result;
    this.section = section;
}
```

+storeInfo(String, int, double, String) : void

```
public void setId(int id){
        this.id = id;
    }
```

+setId(int) : void

```
public int getId(){
    return this.id;
}
```

+getId() : int

```
public void showInfo() {
    System.out.println("Student Name: " + this.name);
    System.out.println("Student ID: " + this.id);
    System.out.println("Result: " + this.result);
    System.out.println("Section: " + this.section);
}
```

+showInfo(): void

```
public static void main(String[] args) {
    Student student1 = new Student();
    Student student2 = new Student();

student1.storeInfo("Alice", 101, 88.5, "A");
student2.storeInfo("Bob", 102, 92.0, "B");

System.out.println("Student 1 Information:");
  student1.showInfo();
System.out.println(); [Blank Line]

System.out.println("Student 2 Information:");
  student2.showInfo();

    }

}
```

+main(String[]args): void

2. (a) এই code টা pets class বানানোর জন্য। এখানে কিছু constructor overloading এবং method overloading use করা হয়েছে।

=>

```
public class Pets {
    public String color;
    public String activity;
    public int weight;
```

Pets class; Instance Variable

```
    public Pets() {
        this.color = "White";
        this.activity = "Sitting";
        this.weight = 0;
    }
```

→ No-argument constructor
default color "White"
activity "Sitting"

```
    public Pets(String color, String activity) {
        this.color = color;
        this.activity = activity;
        this.weight = 0;
    }
}
```

Constructor দুটো color এবং activity set করে

```
public Pets(String color){
    this.color = color;
    this.activity = "Sitting";
    this.weight = 0;
}
```

Constructor এ যাই color set করব, Activity default set করব।

```
public void printPets(){
System.out.println(this.color+" dog is "+this.activity);
}
```

printPets method

```
public void updateInfo(String color){
    this.color = color;
}
```

method overload হবে, colour update হবে।

```
public void updateInfo(int weight){
    this.weight = weight;
  System.out.println(this.color+" dog's weight is now "+
    weight+"kg");
}
```

weight update হবে

```
public void updateInfo(String color, String activity){
    this.color = color;
    this.activity = activity;
}
```

Color ও activity দুইটাকে update করবে।

→ main method এখান থেকে আমরা Pets class এর object create করে output দেখব।

```
public static void main(String[] args){
    Pets dog1 = new Pets();
    Pets dog2 = new Pets("Brown", "Jumping");
    Pets dog3 = new Pets("Black");

    dog1.printPets();
    dog2.printPets();
    dog3.printPets();
```

→ Pet গুলোর Information print করা।

```
    dog2.updateInfo(10);
    dog1.updateInfo("Gray");
    dog3.updateInfo("Cinnamon", "Eating");
```

→ UpdateInfo method use করে Information update করা।

```
    dog1.printPets();
    dog3.printPets();
}
}
```

→ Update এরপর output print করা

Summary:

1. Constructor overloading এর মাধ্যমে pet এর color ও activity আলাদা আলাদাভাবে set করা হয়েছে।
2. Method overloading এর মাধ্যমে color, activity ও weight update করা হয়েছে।
3. Main method এর মাধ্যমে pet গুলোর information print করা হয়েছে।

b) Specify all 4 categories of methods with Proper Java Syntax examples.

=>

1. User Defined Methods: এইগুলো হলো সেই method যেগুলো আমরা নিজের দরকার অনুযায়ী define করি। Java তে instance methods, static methods, abstract methods etc. সহ সব user-defined হতে পারে।

Example:

```
class Calculator{
    public int add(int a, int b){
        return a+b;
    }
}
```

user-defined method to add two numbers

2. Standard Library Methods (SLM): SLM হলো Java এর build-in methods যেগুলো Java Standard Library থেকে আসে। System.out.println(), Math.pow(), String.length().

Ex:

```
public class Main{
    public static void main(String[] args){
        system.out.println("My name is Rohan");
    }
```

3. Overload Methods: Method overloading মানে একই name এর multiple methods থাকা কিন্তু different parameters নিয়ে।

Ex:

```
class Calculator {
    public int add(int a, int b){
        return a+b;
    }
    public int add(int a, int b, int c){
        return a+b+c;
    }
}
```

4. Overridden Methods: Method overriding হলো parent class এর methodকে child class এ redefine করা। Inheritance এর মাধ্যমে subclass parent class এর method কে override করে specific behavior provide করা যায়।

Ex:

```
class Animal {
    public void sound() {
        System.out.println("Animal makes a sound");
    }
}
```

```
class Dog extends Animal {
    public void sound(){
    System.out.println("Dog barks");
}
public static void main(String[] args){
    Animal myDog = new Dog();
    myDog.sound();
    }
}
```

c) (i) Public Box(){}, (ii) Box b1 = new Box(5);
Identify the syntax details of above 2 lines and mention what is being done by which line.

=> Line (i) => (1) Public → access modifier, বাইরে থেকে access করা যাবে
(2) Box() → Constructor → যা object create করার সময় call হয়।
(3) {} → এই brace এর মধ্যে constructor এর body থাকে, এখানে কোনো কোড নাই, মানে constructor টি empty.

Line (ii) => : (1) Box → class এর Name → object creation statement
(2) b1 → একটা variable → Box type এর object store করবে,
(3) new → এই keyword দিয়ে Object create করা হয়,
(4) Box(5) → variable এ 5 দিয়ে constructor কে call করা হচ্ছে যা

দিয়ে Box class এর object বানাচ্ছে।

d) Explanation:—

1. Class A হবে superclass বা parent class. আর Class B এবং Class C হবে তার subclasses.

2. Class B এবং Class C দুইটা আলাদা subclass হবে, যে Class A inherit করবে।

Inheritance Relationship:

* Class A → Parent Class (Superclass)
* Class B and Class C → Child Classes (Subclass), যা Class A থেকে inherit করবে।

e) Already solved.