CSE 201 ADVANCED PROGRAMMING

MidSem Examination [2021-22] [Total 20 marks, Duration: 1hr]

[Section A]

Answer any SEVEN of the following. Don't spend more than 3 minutes on any of these questions. **[7x1=7 marks]**

Check for the points mentioned in the answers.

If all of them are there, give one mark.

If they are partially there, give half mark.

If they are not there, but some other correct points are there, give half mark.

If none of the points made are correct, give zero marks.

Q1) Why and where do we need @Override annotation?

It is required to warn developers at the compile time about the error they made in the signature of overriding function. It should be mentioned just above the overriding function.

Q2) In what all ways can we overload a function?

We can overload a function by changing the number of arguments and their types.

Q3) Name the four types of access modifiers. Discuss how they modify accessibility.

Public: Accessible everywhere

Private: Accessible only within the class

Protected: Accessible within the package and the derived class outside the package.

Default: Accessibility only within the package.

Q4) What is the difference between composition and association class relationships?

Composition relationship: The instance variables of another type is initialized with an object created in the class.

Association relationship: The instance variables of another type are initialized with the reference of an object created outside the class.

Q5) What is the difference between dependency and association class relationships?

Dependency Relatonship: The reference of an object of another type is passed to the function for initializing an instance variable.

Association Relationship: The reference of an object of another type is passed to the function for initializing a local variable.

Q6) How does a setter function help in Encapsulation?

Setter function controls how a private member can be set based on the criteria the setter function might be having.

Q7) Differentiate between static and instance variables in a class.

Static variables belong to the class whereas instance variables belong to the objects.

Static variables are accessible to both static and non-static methods, but instance variables are accessible to only non-static methods.

Q8) Differentiate between static and non-static methods in a class.

Static methods belong to the class whereas non-static methods belong to the objects.

Static methods can be called using both the class and an object whereas non-static methods can be called using only the object.

Static methods can access only static variables whereas non-static methods can access both static and non-static variables.

Q9) In what all ways can we initialize instance variables of a class?

Right at the time of declaration.

Inside an initialization block.

Inside a constructor.

Q10) What do you mean by reference and object types of a variable? [NOTE: They are also known as declared and actual types, respectively]

Reference type means the type used to declare the variable whereas the object type means the type of the object the variable is referring to.

[Section B]

Answer any TWO of the following. Don't spend more than 6 minutes on any of these questions. **[2x3=6 marks]**

Q1) When does a compiler link a function to a function call? When it doesn't, what links them and how?

A compiler checks whether the method defined in the reference type class of the calling variable has any of the three keywords: static, private and final. If there, it will call the function at the compile time itself; otherwise it leaves it to the JVM, which checks the method in the object type class of the calling variable.

Compiler using reference type: 0.5 marks

Mentioned the three keywords: 1.5 mark

Mentioned JVM: 0.5 marks

JVM using object type: 0.5 marks

Q2) Write a program to illustrate the importance of super keyword in avoiding null pointer exception.

Ensure that the super function is used to pass an object's reference in the child class to initialize a private instance variable of the parent class, which must be having a constructor to call it through super function.

Reference passed through super function in child class: 1 mark

A constructor is there in the parent class to initialize an instance variable: 1 mark

The instance variable being initialized is private: 1 mark

Q3) In what all ways can we make a class immutable? Give an example of an immutable class.

Check for at least 4 of the following points [4x0.5=2 marks]

- *Don't provide any methods that modify the object's state.*
- *Make all fields private. (ensure encapsulation)*
- *Make all fields final.*
- *Ensure exclusive access to any mutable object fields.*
- *Don't let a client get a reference to a field that is a mutable object (don't allow any mutable representation exposure.)*
- *Ensure that the class cannot be extended.*
  [0.5 marks for each]

For the example, award 0.25 marks for each attempt to make the class immutable. And cap the total to 1 mark. [0.25x4=1 mark]

[Section C]

Answer any ONE of the following. **[1x7=7 marks]**

Q1) Write a program that illustrates all three kinds of class relationships.

1) One of the classes must have an instance variable of another class, which should be initialized through reference of an object created within the class. [2 marks]
2) One of the classes must have an instance variable of another class, which should be initialized through reference of an object created outside the class. [2 marks]
3) One of the functions of a class must have an argument of another class, and that function does something with the argument received (except initializing the instance variables). [2 marks]
4) If no errors. [1 mark]

Q2) Identify errors in the following code and rectify them:

```
interface Animal{
    public void eat(Animal A);
    public void move(Animal A);
}

abstract class Bird implements Animal{
    abstract public void eat(Animal A);
    abstract public void move(Animal A);
}

class Parrot extends Bird{
    protected String name;
    Parrot(String name){
        name=name;
    }
    @Override
    public void eat(Bird A){
        System.out.println(this.name+ " eats with " + ((Parrot)A).name);
    }
    @Override
    static void move(Animal A) {
        System.out.println(this.name+ " flies with " + ((Parrot)A).name);
    }
}

class Crow extends Parrot{
    Crow(String name){
        super();
    }
    @Override
    public void eat(Animal A) {
        System.out.println(super.name+ " eats with " + ((Parrot)A).name);
    }
    @Override
    public void move(Animal A) {
        System.out.println(super.name+ " flies with " + ((Parrot)A).name);
    }
}

class Main2 {
    public static void main2(String[] args) {
        Animal Ap1=new Animal("parrot1");
        Bird Ac1 = new Crow("crow1");
        Parrot Ap2 = new Parrot("parrot2");
        Crow Ac2 = new Crow("crow2");
        Bird [] birds = {Ap1,Ac1,Ap2,Ac2};
        birds[0].eat(birds[1]);
        birds[0].move(birds[2]);
        birds[2].eat(birds[3]);
        birds[3].move(birds[1]);
    }
}
```

Expected Output:

parrot1 eats with crow1

parrot1 flies with parrot2

parrot2 eats with crow2

crow2 flies with crow1

```
interface Animal{
    public void eat(Animal A);
    public void move(Animal A);
}

abstract class Bird implements Animal{
    abstract public void eat(Animal A);
    abstract public void move(Animal A);
}

class Parrot extends Bird{
    protected String name;
    Parrot(String name){
        this.name=name;//added 'this.' [1 mark]
    }
    @Override
    public void eat(Animal A){//Bird->Animal [1 mark]
        System.out.println(this.name+ " eats with " +
((Parrot)A).name);
    }
    @Override
    public void move(Animal A) {//static [1 mark]
        System.out.println(this.name+ " flies with " +
((Parrot)A).name);
    }
}

class Crow extends Parrot{
    Crow(String name){
        super(name);//added name as an argument [1 mark]
    }
```

```
    @Override
    public void eat(Animal A) {
        System.out.println(super.name+ " eats with " +
((Parrot)A).name);
    }
    @Override
    public void move(Animal A) {
        System.out.println(super.name+ " flies with " +
((Parrot)A).name);
    }
}

class Main2 {
    public static void main(String[] args) {//main2->main [1
mark]
        Animal Ap1=new Parrot("parrot1");//Animal->Parrot [1
mark]
        Bird Ac1 = new Crow("crow1");
        Parrot Ap2 = new Parrot("parrot2");
        Crow Ac2 = new Crow("crow2");
        Bird [] birds = {(Bird)Ap1,Ac1,Ap2,Ac2};//casted Ap1
to Bird [1 mark]
        birds[0].eat(birds[1]);
        birds[0].move(birds[2]);
        birds[2].eat(birds[3]);
        birds[3].move(birds[1]);
    }
}
```