CSC215 Marty Martin
2:00 PM - 3:20 PM May 2024

Practice for the exam

Question A.1

How is a programmer different from a coder?

Solution

A ***programmer*** is a professional who designs, develops, tests, and maintains complex software applications. They are involved in problem-solving, algorithm design, and often work on integrating various systems and technologies. A ***coder***, on the other hand, primarily focuses on writing code. Their role is often more specific and less broad, mainly translating algorithms and program designs into executable code without necessarily being involved in the higher-level aspects of software development.

Question A.2

Please list and explain the general components of a Java class that represents lower-division Computer Science students.

Solution

A Java class representing lower-division Computer Science students might include the following components:

- ***Attributes***: Variables within a class such as ***studentId*** (String), ***name*** (String), ***email*** (String), and ***currentGPA*** (double).
- ***Methods***: Functions like ***enrollInCourse()***, ***calculateGPA()***, and ***updateEmail()*** that operate on the attributes.
- ***Constructor***: Special method used to create instances, initializing attributes like ID, name, and email.
- ***Access Modifiers***: Define visibility of components, using ‘private‘ for attributes and ‘public‘ for methods to ensure encapsulation.

Question A.3

Please list and explain 3 differences between static variables and non-static variables.

Solution

- ***Scope***: ***Static variables*** are shared among all instances of a class, while ***non-static variables*** (instance variables) are unique to each instance.
- ***Memory Allocation***: ***Static variables*** are allocated once the class is loaded, whereas ***non-static variables*** are allocated each time an object is instantiated.
- ***Access***: ***Static variables*** can be accessed directly using the class name, unlike ***non-static variables*** which require an instance.

Question A.4

Please explain Inheritance and explain Polymorphism. Explain the connection between these 2 OOP concepts. What are the OOP pillars?

Solution

- ***Inheritance*** is a mechanism where a new class (subclass) derives properties and behaviors from an existing class (superclass). It allows code reusability and hierarchical classification.
- ***Polymorphism*** allows using a single interface for entities of different types. It enables a method to perform different tasks based on the object that invokes it.
- **Connection**: Inheritance provides the hierarchical relationship necessary for polymorphism. A subclass can override or extend the functionality of its superclass, enabling polymorphic behavior.
- **OOP Pillars**: The four pillars of Object-Oriented Programming (OOP) are ***Encapsulation***, ***Inheritance***, ***Polymorphism***, and ***Abstraction***.

Question A.5

Please list and explain 3 differences between Up-casting and Down-casting.

Solution

- ***Direction***: ***Up-casting*** is casting a subclass to its superclass, reducing visibility of methods and properties. ***Down-casting*** is casting a superclass to its subclass, increasing visibility but requiring explicit casting.
- ***Safety***: ***Up-casting*** is always safe as every subclass is a type of its superclass. ***Down-casting*** can lead to a ***ClassCastException*** if not properly handled or if the object is not of the target type.

- ***Usage Context***: ***Up-casting*** is often used to generalize a type for broader method applicability. ***Down-casting*** is used when specific subclass functionalities need to be accessed after an object has been treated generically.

Question A.6

Please explain how the below operator and methods work. What are the differences among them? 1. equality operator == 2. equals method 3. compareTo method 4. Differences

Solution

- ***Equality Operator (==)***: This operator checks if two references point to the exact same object in memory. It's about reference equality, not the content of the objects.
- ***Equals Method***: This is a method in the ***Object*** class that can be overridden. By default, it behaves like ***==***, but when overridden (like in ***String*** or ***List***), it can compare the contents of two objects.
- ***CompareTo Method***: Found in the ***Comparable*** interface, this method compares two objects for order. It returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.
- ***Differences***: The ***== operator*** compares references, the ***equals method*** compares object content when properly overridden, and the ***compareTo method*** provides ordering information about the objects.

Question B.1

Please code a complete Java program: VacationAgency.

- Your program prompts users to enter their favorite Summer vacation destination choice.
- Then the program prints a recommendation on how to get there.
- It is OK to assume that users will enter a valid location as their favorite Summer vacation destination.
- This program must have at least 3 methods (1 of them is the main method).
- A sample run of the program (think Google Maps):

```
Enter your Summer vacation destination: Phu Quoc Island
Flight Vietnamese Airlines!
```

Solution

Here's a Java program named `VacationAgency`:

```java
import java.util.Scanner;

public class VacationAgency {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your Summer vacation destination: ");
        String destination = scanner.nextLine();
        printTransportRecommendation(destination);
    }

    // Method to recommend transport based on the destination
    public static void printTransportRecommendation(String destination) {
        if (destination.equalsIgnoreCase("Phu Quoc Island")) {
            System.out.println("Flight: Vietnamese Airlines!");
        } else if (destination.equalsIgnoreCase("Paris")) {
            System.out.println("Flight: Air France!");
        } else {
            System.out.println("Train or Local Airlines are recommended.");
        }
    }

    // Additional method for future expansion or specific recommendations
    public static void additionalRecommendations(String destination) {
        // This can be used for additional advice like hotels, etc.
        System.out.println("Don't forget to enjoy the local cuisine at " +
           destination + "!");
    }
}
```

Question B.2

Please choose a real-life entity then write a Java class to represent it. The entity must be part of San Francisco State University.

- What is that entity?
- Why do you think it is suitable to be a Java class?
- Please code the class. Your class should have data fields, constructors, and methods.

Solution

The chosen entity is a **Student** from San Francisco State University.

Reasons for Suitability: A Student is suitable as a Java class because it encapsulates specific attributes and behaviors like student ID, name, email, and GPA operations, which can be modeled computationally.

```java
public class Student {
    private String studentId;
    private String name;
    private String email;
    private double gpa;
    private static int count = 0; // To keep track of the number of Student
        objects created

    // Constructor
    public Student(String studentId, String name, String email, double gpa) {
        this.studentId = studentId;
        this.name = name;
        this.email = email;
        this.gpa = gpa;
        count++; // Increment count whenever a new student is created
    }

    // Methods
    public void updateEmail(String newEmail) {
        this.email = newEmail;
    }

    public void printDetails() {
        System.out.println("Student ID: " + studentId + ", Name: " + name + ",
            Email: " + email + ", GPA: " + gpa);
    }

    public static int getStudentCount() {
        return count;
    }
}
```

Question B.3

Please write code to add another data field to your class to keep track of the number of objects created.

- Please explain in detail how the data field should be used. Provide code to demonstrate and support your explanation.
- Then explain in detail why your code should work properly. Provide code to demonstrate and support your explanation.

Solution

The data field `count` is used to keep track of the number of `Student` objects created. This is a static field, meaning it is shared among all instances of the class and not specific to any single object.

Detailed Explanation with Code:

```
public static void main(String[] args) {
    Student s1 = new Student("S001", "Alice", "alice@example.com", 3.7);
    Student s2 = new Student("S002", "Bob", "bob@example.com", 3.5);

    System.out.println("Total students: " + Student.getStudentCount());
}
```

This will print ”Total students: 2” as two `Student` objects are created, and each constructor call increases the `count` by one.

Question B.4

Write a no-argument constructor, a two-argument constructor, and a four-argument constructor for your class.

- Code instructions to create 3 objects using all constructors above.

Solution

Here are the requested constructors for the `Student` class:

```java
// No-argument constructor
public Student() {
    this.studentId = "Unknown";
    this.name = "Unknown";
    this.email = "noemail@example.com";
    this.gpa = 0.0;
    count++;
}

// Two-argument constructor
public Student(String studentId, String name) {
    this.studentId = studentId;
    this.name = name;
    this.email = "noemail@example.com";
    this.gpa = 0.0;
    count++;
}

// Four-argument constructor
public Student(String studentId, String name, String email, double gpa) {
    this.studentId = studentId;
    this.name = name;
    this.email = email;
    this.gpa = gpa;
    count++;
}

// Creating objects using all constructors
public static void main(String[] args) {
    Student s1 = new Student();
    Student s2 = new Student("S003", "Charlie");
    Student s3 = new Student("S004", "Diana", "diana@example.com", 3.9);
}
```

Question B.5

Singer and Student are subclasses of Person.

- Please write the header line of each class. Then please explain in detail what happens when a Singer constructor is called. Please explain both the explicit and implicit code.
- Please write code to demonstrate a subclass reference to reference a subclass object.
- Please write code to demonstrate a superclass reference to reference a subclass object.
- Please write code to demonstrate up-casting.
- Please write code to demonstrate down-casting.

Solution

1. **Header Line of Each Class:**

```
public class Person {
   // Base class for Singer and Student
}

public class Singer extends Person {
   // Singer-specific code here
}

public class Student extends Person {
   // Student-specific code here
}
```

Explanation of Singer Constructor: When a `Singer` constructor is called, Java first implicitly calls the constructor of the superclass (`Person`). If an explicit call to `super()` with parameters isn't provided, Java will call the no-argument constructor of `Person`. This ensures that the initialization of the parent class (`Person`) is done before the child class (`Singer`).

2. **Subclass Reference to Reference a Subclass Object:**

```
Singer singer1 = new Singer();
```

3. **Superclass Reference to Reference a Subclass Object:**

```
Person student1 = new Student();
```

4. **Up-casting Code:**

```
Singer singer1 = new Singer();
Person person1 = singer1; // Up-casting Singer to Person
```

5. **Down-casting Code:**

```java
Person person2 = new Singer();
Singer singer2 = (Singer) person2; // Down-casting Person to Singer
```

Question B.6

Please override the method compareTo to compare Phone objects.

Solution

To override the `compareTo` method for comparing `Phone` objects, ensure your `Phone` class implements the `Comparable` interface. Here's how you can do it:

```java
public class Phone implements Comparable<Phone> {
    private String model;
    private double price;

    // Constructor
    public Phone(String model, double price) {
        this.model = model;
        this.price = price;
    }

    // Override compareTo to compare based on price
    @Override
    public int compareTo(Phone other) {
        return Double.compare(this.price, other.price);
    }

    // Example usage
    public static void main(String[] args) {
        Phone phone1 = new Phone("Model A", 499.99);
        Phone phone2 = new Phone("Model B", 599.99);

        System.out.println(phone1.compareTo(phone2)); // Will print a negative
             value since phone1 is cheaper
    }
}
```