

Class Activity4

CTU – Training Solutions
UID – User Interface Development
Software Development
SWD412
Week 4

Complete the following Class activity Java Exercises:

1. Polymorphism

- Exercise: Create a base class Shape with a method area().
- Then create two subclasses Rectangle and Circle that override the area() method to calculate the area for each shape.
- Write a main method to demonstrate polymorphism by calling the area() method on an array of Shape references.

```
public class Polymorphism {  
  
    // Base Shape class  
    static class Shape {  
        // Method to calculate area (to be overridden by subclasses)  
        public double area() {  
            return 0; // Default implementation (could also be abstract)  
        }  
    }  
  
    // Subclass Rectangle that extends Shape  
    static class Rectangle extends Shape {  
        private double length;  
        private double width;  
  
        // Constructor  
        public Rectangle(double length, double width) {  
            this.length = length;  
            this.width = width;  
        }  
  
        // Override the area() method to calculate the area of a rectangle
```

```

        @Override
        public double area() {
            return Math.PI * radius * radius;
        }
    }

    // Main method to demonstrate polymorphism
    public static void main(String[] args) {
        // Creates an array of Shape references
        Shape[] shapes = new Shape[2];

        // Initialize the array with Rectangle and Circle objects
        shapes[0] = new Rectangle( length:5, width:7);
        shapes[1] = new Circle( radius:3);

        // Loop through the array and call the area() method on each shape in the program
        for (Shape shape : shapes) {
            System.out.println("Area: " + shape.area());
        }
    }
}

```

```

100%
Area: 35.0
Area: 28.274333882308138
BUILD SUCCESSFUL (total time: 0 seconds)
|

```

2. Inheritance

- Exercise: Create a base class Vehicle with properties like make and model, and a method displayInfo().
- Then create a subclass Car that inherits from Vehicle and adds a property numberOfDoors. Override the displayInfo() method in Car to include the number of doors.
- Write a main method to demonstrate inheritance.

```

/*
 * @author Dian
 */
public class Vehicle {
    // Properties of Vehicle
    String make;
    String model;

    // Constructor for Vehicle
    public Vehicle(String make, String model) {
        this.make = make;
        this.model = model;
    }

    // Method to display Vehicle information
    public void displayInfo() {
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
    }
}

```

```

    * @author Dian
    */
    // Subclass Car that inherits from Vehicle
    public class Car extends Vehicle {
        // Property specific to Car
        int numberOfDoors;

        // Constructor for Car
        public Car(String make, String model, int numberOfDoors) {
            // Call the constructor of the superclass Vehicle
            super(make, model);
            this.numberOfDoors = numberOfDoors;
        }

        // Override the displayInfo() method to include numberOfDoors
        @Override
        public void displayInfo() {
            // Call the superclass's displayInfo() method
            super.displayInfo();
            System.out.println("Number of Doors: " + numberOfDoors);
        }
    }

    // Main class to demonstrate inheritance
    public class Inheritance {
        public static void main(String[] args) {
            // Create an instance of Vehicle
            Vehicle vehicle = new Vehicle("Toyota", "Camry");
            System.out.println("Vehicle Information:");
            vehicle.displayInfo();

            // Create an instance of Car
            Car car = new Car("Honda", "Civic", 4);
            System.out.println("Car Information:");
            car.displayInfo();
        }
    }

```

Vehicle Information:

Make: Toyota

Model: Camry

Car Information:

Make: Honda

Model: Civic

Number of Doors: 4

BUILD SUCCESSFUL (total time: 0 seconds)

3. Encapsulation

- Exercise: Create a class BankAccount with private fields for accountNumber and balance.
- Provide public methods to deposit and withdraw funds, as well as a method to check the balance. Write a main method to demonstrate encapsulation.

```
public class Bankaccount {  
    // Private fields  
    private String accountNumber;  
    private double balance;  
  
    // Constructor to initialize accountNumber and balance  
    public Bankaccount(String accountNumber, double initialBalance) {  
        this.accountNumber = accountNumber;  
        this.balance = initialBalance;  
    }  
  
    // Public method to deposit funds  
    public void deposit(double amount) {  
        if (amount > 0) {  
            balance += amount;  
            System.out.println("Deposited: $" + amount);  
        } else {  
            System.out.println(x: "Deposit amount must be positive.");  
        }  
    }  
  
    // Public method to withdraw funds  
    public void withdraw(double amount) {  
        if (amount > 0) {  
            if (amount <= balance) {  
                balance -= amount;  
                System.out.println("Withdrew: $" + amount);  
            } else {  
                System.out.println(x: "Insufficient funds.");  
            }  
        } else {  
            System.out.println(x: "Withdrawal amount must be positive.");  
        }  
    }  
  
    // Public method to check the balance  
    public double getBalance() {  
        return balance;  
    }  
  
    // Public method to get the account number (if needed)  
    public String getAccountNumber() {  
        return accountNumber;  
    }  
}
```

```

// Main method to demonstrate encapsulation
public static void main(String[] args) {
    // Create a BankAccount object
    Bankaccount account = new Bankaccount( accountNumber: "123456789", initialBalance: 1000.00);

    // Display initial balance
    System.out.println("Initial Balance: $" + account.getBalance());

    // Deposit funds
    account.deposit( amount: 500.00);

    // Withdraw funds
    account.withdraw( amount: 200.00);

    // Try to withdraw more than the balance
    account.withdraw( amount: 2000.00);

    // Check final balance
    System.out.println("Final Balance: $" + account.getBalance());
}
}

```

4. Constructors and Methods

- Exercise: Create a class Book with properties for title, author, and price.
- Include a constructor to initialize these properties and methods to display book details and apply a discount.
- Write a main method to demonstrate the usage of constructors and methods.

```

public class Books {
    private String title;
    private String author;
    private double price;

    // Constructor to initialize properties
    public Books(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    // Method to display book details
    public void displayDetails() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Price: $" + price);
    }
}

```

```

// Method to apply a discount
public void applyDiscount(double discountPercentage) {
    if (discountPercentage > 0 && discountPercentage <= 100) {
        double discountAmount = price * (discountPercentage / 100);
        price -= discountAmount;
        System.out.println("Discount of " + discountPercentage + "% applied.");
    } else {
        System.out.println(x: "Invalid discount percentage.");
    }
}

// Main method to demonstrate usage
public static void main(String[] args) {
    // Create a Book object
    Books book = new Books( title: "The Great Gatsby", author: "F. Scott Fitzgerald", price: 15.99);

    // Display book details
    System.out.println(x: "Book Details:");
    book.displayDetails();

    // Apply a discount
    book.applyDiscount( discountPercentage: 10);

    // Display book details after discount
    System.out.println(x: "\nBook Details After Discount:");
    book.displayDetails();
}

```

Revision

Encapsulation

Encapsulation is the bundling of data (attributes) and methods (functions) that operate on the data into a single unit or class. It restricts direct access to some of the object's components, which is a way of preventing accidental interference and misuse of the data.

Inheritance

Inheritance allows a class (child class) to inherit attributes and methods from another class (parent class). It helps in code reusability and creating a relationship between classes.

Constructor

A constructor is a special method that is automatically called when an object of a class is created. It's typically used to initialize the object's attributes.

Polymorphism

Polymorphism refers to the ability of different classes to be treated as instances of the same class through inheritance. It also refers to the ability to use a single method name to perform different tasks based on the object it is acting upon.

Revision question

1. What is Java?

Java is a high-level, class-based, object-oriented programming language designed to have as few implementation dependencies as possible.

It is widely used for building large-scale enterprise applications, mobile applications (especially Android apps), web-based applications, and more.

2. Why do we use NetBeans?

NetBeans is an Integrated Development Environment (IDE) used for developing Java applications.

It provides a user-friendly interface, tools for code editing, debugging, compiling, and version control, among other features.

3. How does java run on your pc?

Java code is first compiled into bytecode by the Java compiler.

This bytecode is platform-independent and can run on any system with a JVM (Java Virtual Machine).

4. Is java relevant in today's industry?

It is widely used in various domains, including enterprise software development, Android mobile development, web applications, cloud-based services, and even big data technologies like Hadoop. Java's robustness, security features, scalability, and vast ecosystem make it a go-to language for many developers and organizations.

5. Where is java most widely used?

- ☒ **Enterprise applications:** Java is a standard for building large-scale, reliable, and secure business applications.
- ☒ **Android development:** Java is the primary language for Android app development.
- ☒ **Web applications:** Java is used in web servers and backend development.
- ☒ **Big Data technologies:** Java is used in big data processing frameworks like Apache Hadoop.
- ☒ **Financial services:** Java is often used for developing trading systems, banking applications, and more.

6. What is a root/project folder?


A root/project folder is the top-level directory in a project's file structure where all the project files and subdirectories are stored. This folder contains source code files, configuration files, libraries, and other resources needed for the project.

7. Provide a screenshot of you creating a 'Hello, World!' application


```
run:
Hello, World!
BUILD SUCCESSFUL (total time: 0 seconds)
```


main.java x

SourceHistory







1



```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit th
4  */
5
6  /**
7   *
8   * @author Dian
9   */
10 public class main {
11     public static void main(String[] args) {
12         System.out.println("Hello, World!");
13     }
14 }
```

main >

Output - JavaApplication2 (run) x

```
run:
Hello, World!
BUILD SUCCESSFUL (total time: 0 seconds)
```

1. Explain what are variables used for?

Variables are used to store data that can be referenced and manipulated in a program. They act as containers for values that your program can use, such as numbers, text, or more complex data types. Variables are essential for storing inputs, outputs, intermediate results, and any other data that your program needs to process.

2. Which datatypes are used more commonly in each application?

3. **int**: For integer numbers (e.g., `int age = 25;`).
4. **double/float**: For floating-point numbers (e.g., `double price = 19.99;`).
5. **char**: For single characters (e.g., `char letter = 'A';`).
6. **String**: For sequences of characters (e.g., `String name = "Alice";`).
7. **boolean**: For true/false values (e.g., `boolean isActive = true;`).
8. **Array/List**: For collections of items (e.g., `int[] numbers = {1, 2, 3};`).

9. Why do we import a print method?

In Java, the `System.out.println()` method is used to print text to the console. It is part of the `java.lang` package, which is automatically imported. In other languages, you may need to import specific libraries or modules to use print methods or functions.

10. What is the difference between a method and function?

Method: In object-oriented programming, a method is a function that is associated with an object or class. It operates on the data contained within that object.

Function: A function is a block of code that performs a specific task. It can exist independently (in some programming languages) or be associated with an object (as a method). In Java, all functions are methods because they must be associated with a class or object.

11. Where are classes used and where will objects be used?

Classes are used to define the blueprint for creating objects. They encapsulate data for the object and define its behavior through methods. You define a class when you want to create a new type of object with specific properties and behaviors.

Objects are instances of classes. They are used when you need to create and manipulate data according to the blueprint provided by a class. An object represents a specific entity in your application, with its own state and behavior based on its class.