National University of Computer and Emerging Sciences

Karachi Campus

Object Oriented Programming (CS1004)

Final Exam

Total Time (Hrs): 3

Total Marks: 100

Total Questions: 06

Date: May 20th 2024

Course Instructor(s)

Mr. Basit, Mr. Minhal, Ms.Sumaiya, Ms.Sobia, Ms.Abeeha, Ms.Bakhtawer, Ms.Abeer, Ms.Atiya, Ms.Rafia

_____ _____ _____

Roll No Section Student Signature

Do not write below this line

Attempt all the questions.

CLO #1: Discuss knowledge of underlying concepts of object-oriented paradigm like abstraction, encapsulation, polymorphism, inheritance etc

Q1: Write short answers (2-3 lines) for the following questions: **[10 minutes, 1*5=5 marks]**

a) What is the difference between compile-time error and runtime error? Which types of errors can be dealt with using exceptions?

Exceptions can be handled during run-time whereas errors cannot be because exceptions occur due to some unexpected conditions during run-time whereas about errors compiler is sure and tells about them during compile-time

b) Which principle of object-oriented programming suits in the below scenarios?
   1. A bank vault with multiple layers of security protecting valuable assets. Encapsulation
   2. A car with different driving modes for different road conditions.Polymorphism

c) Write an advantage of using overloading instead of generics in JAVA? Overloading allows the programmer to write more specific code for each data type.

d) Can a protected function of Base class access the data members of a derived class? No

e) If class C inherits class B and class B inherits class A, then Class C object can be upcasted to object of either class A or B? Yes

CLO #3: Illustrate Object-Oriented design artifacts and their mapping to Object-Oriented Programming using JAVA

Q2: Predict the output of the following code snippets. If there are any errors, provide a proper justification. Exclude errors related to libraries. **[25 minutes, 2*5=10 marks]**

A.
```
class Test {
  Test() { System.out.println("Hello from Test()");}
  } A a;
public class Main { public static void main(String[] args) {
   System.out.println("Main Started");}
```

B.
```
class Test {
```

```
   public Test() {
      System.out.println("Constructing an object of Test");}
    protected void finalize() {
      System.out.println("Destructing an object of Test");}}
public class Main {
   public static void main(String[] args) {
      try {Test t1 = new Test(); throw new Exception("10");}
catch (Exception e) {System.out.println("Caught " + e.getMessage());}}
```

Constructing an object of Test
Caught 10

C.
```
class A <T>{
          void fun(final T x)
          {
             static int count = 0;
            System.out.println( "x = "  + x + " count = " +count);
             ++count;
             return;
          }
public class Main {
   public static void main(String[] args) {
      Fun.fun(1);
      System.out.println();
      Fun.fun(1);
      System.out.println();
      Fun.fun(1.1);
      System.out.println();
   }
```

D.
```
public class Main {
            public static void main(String[] args) {
               try {
                  System.out.println("Throwing Block");
                  throw new CharException('x');
               } catch (IntException e) {
                  System.out.println("Catching Int: " + e.getValue());
               } catch (CharException e) {
                  System.out.println("Catching Char: " + e.getValue());
               } catch (Exception e) {
                  System.out.println("Catching Default");
               }
            }
          }
```

E.
```
public class Main {
   static char foo(int i) throws Exception {
      if (i % 3 != 0)
         return 't';
```

```
        throw new Exception("o");
    }

    public static void main(String[] args) {
        char c = 'c';
        try {
            for (int i = 1;; i++) {
                c = foo(i);
                System.out.println(c);
            }
        } catch (Exception ex) {
            System.out.println(c);
            System.out.println(ex.getMessage());
        }
    }
}
```

t
t
t
o

CLO #3: Illustrate Object-Oriented design artifacts and their mapping to Object-Oriented Programming using JAVA

Q3: Complete the JAVA code for an Object-Oriented Programming (OOP) scenario.

[25 minutes, 5*2=10 marks]

A. For the given class, you are required to create a specialized template that manages computations specifically when both arrays are characters with a size of 10. Overload the function so that it returns a string containing all elements of arr1 followed by all elements of arr2.

```
public class A<T extends Number> {
    private T[] arr1;
    private T[] arr2;

    public A(T[] arr1, T[] arr2) {
        this.arr1 = arr1;
        this.arr2 = arr2;
    }

    public T[] AddArrays(T[]arr1 , T[] arr2) {
        if (arr1.length != 10 || arr2.length != 10) {
            return null; // Arrays must be of size 10
        }

        T[] result = (T[]) new Number[10];
        for (int i = 0; i < 10; i++) {
            double sum = arr1[i].doubleValue() + arr2[i].doubleValue();

            return result;
```

```
  }
}public static void main(String[] args) {
      Character[] arr1Char = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
      Character[] arr2Char = {'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't'};
      myClass = new MyClass<>(arr1Double, arr2Double);
      Double[] resultDouble = myClass.concatenateArrays(arr1Double, arr2Double);
      if (resultDouble != null) {
         System.out.print("\nResult of addition (Double): ");
         for (int i = 0; i < 10; i++) {
            System.out.print(resultDouble[i] + " ");
         }
      } else {
         System.out.println("\nArrays must be of size 10");
      }
```

Solution:

```
public class A<T> {
   private T[] arr1;
   private T[] arr2;

   public A(T[] arr1, T[] arr2) {
      this.arr1 = arr1;
      this.arr2 = arr2;
   }

   public static Number[] addNumberArrays(Number[] arr1, Number[] arr2) {
      if (arr1.length != 10 || arr2.length != 10) {
         return null; // Arrays must be of size 10
      }

      Number[] result = new Number[10];
      for (int i = 0; i < 10; i++) {
         double sum = arr1[i].doubleValue() + arr2[i].doubleValue();
         result[i] = sum;
      }

      return result;
   }

   public static Character[] addCharacterArrays(Character[] arr1, Character[] arr2) {
      if (arr1.length != 10 || arr2.length != 10) {
         return null; // Arrays must be of size 10
      }

      Character[] result = new Character[10];
      for (int i = 0; i < 10; i++) {
         char sum = (char) (arr1[i] + arr2[i]);
         result[i] = sum;
      }
```

```
        return result;
    }

    public static void main(String[] args) {
        // Example with Double arrays
        Double[] arr1Double = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0};
        Double[] arr2Double = {11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0};

        Number[] resultDouble = A.addNumberArrays(arr1Double, arr2Double);

        if (resultDouble != null) {
            System.out.print("Result of addition (Double): ");
            for (int i = 0; i < 10; i++) {
                System.out.print(resultDouble[i] + " ");
            }
        } else {
            System.out.println("Arrays must be of size 10");
        }

        // Example with Character arrays
        Character[] arr1Char = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
        Character[] arr2Char = {'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't'};

        Character[] resultChar = A.addCharacterArrays(arr1Char, arr2Char);

        if (resultChar != null) {
            System.out.print("\nResult of addition (Character): ");
            for (int i = 0; i < 10; i++) {
                System.out.print(resultChar[i] + " ");
            }
        } else {
            System.out.println("\nArrays must be of size 10");
        }
    }
}
```

If you remove T extends Number from the class definition, T can be any type, not just numeric types. This means that the generic AddArrays method can no longer assume that T has the doubleValue() method, which is specific to the Number class. To handle both numeric arrays and character arrays with a more flexible approach, you could overload methods without relying on the Number type.

Here's how you can define the class and methods to handle both numeric and character arrays:

Remove the type constraint from the class.

Use separate methods to handle numeric operations and character operations.The addNumberArrays method is static and specifically handles arrays of Number types (e.g., Double, Integer). It converts the numbers to double for addition.
The addCharacterArrays method is static and handles arrays of Character types. It casts the sum of two characters to a char.

The main method demonstrates the usage of both methods with Double arrays and Character arrays. This approach separates the handling of numeric and character arrays into distinct methods, avoiding the need to use generics constrained to Number.

B. Complete the given code below in such a way that when we run this code, a similar kind of output is stored in the file.

outfile.txt:
ID = 1, Name = 0001
ID = 2, Name = 0002

```
class Test {
   int ID;
   string name;
   static int genID;
public:
//code is required here
};


main() {
   Test t1, t2;
   t1.Add(t1,"outfile.txt");
   t2 .Add(t2,"outfile.txt");
}
```

Solution (1)

```
import java.io.*;

class Test implements Serializable {
   int ID;
   String name;
   static int genID = 0;

   // Constructor to initialize Test object
   public Test(String name) {
      this.ID = ++genID;
      this.name = name;
   }

   // Method to add Test object to a file using serialization
   public static void Add(Test obj, String fileName) {
      try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fileName, true))) {
         oos.writeObject(obj);
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
```

```
    // Method to read Test objects from a file
    public static void readObjects(String fileName) {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileName))) {
            while (true) {
                try {
                    Test obj = (Test) ois.readObject();
                    System.out.println("ID: " + obj.ID + ", Name: " + obj.name);
                } catch (EOFException eof) {
                    break;
                }
            }
        } catch (IOException) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        // Creating Test objects
        Test test1 = new Test("Alice");
        Test test2 = new Test("Bob");

        // File to store serialized objects
        String fileName = "test_output.ser";

        // Writing objects to the file
        Add(test1, fileName);
        Add(test2, fileName);

        System.out.println("Objects have been written to the file: " + fileName);

        // Reading objects from the file
        System.out.println("Reading objects from the file:");
        readObjects(fileName);
    }
}
```

Sloution (2)

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

class Test {
    int ID;
    String name;
    static int genID = 0;

    // Constructor to initialize Test object
    public Test(String name) {
        this.ID = ++genID;
```

```
        this.name = name;
    }

    // Method to add Test object to a file
    public static void Add(Test obj, String fileName) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName, true))) {
            writer.write("ID: " + obj.ID + ", Name: " + obj.name);
            writer.newLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        // Creating Test objects
        Test test1 = new Test("Alice");
        Test test2 = new Test("Bob");

        // Writing objects to the file
        String fileName = "test_output.txt";
        Add(test1, fileName);
        Add(test2, fileName);

        System.out.println("Objects have been written to the file: " + fileName);
    }
}
```

CLO #4: Design and assess small and medium scale JAVA programs using object-oriented programming principles

---

Q4: Sui Southern Gas Company (SSGC) is one of the leading natural gas utility companies in Pakistan, responsible for distributing natural gas to various regions. As part of its operations, SSGC maintains a fleet of vehicles for meter reading, pipeline inspection, and maintenance tasks. To enhance operational efficiency and ensure proper management of the fleet, SSGC has decided to implement a Fleet Management System (FMS). This system will track vehicle locations, monitor fuel consumption, analyze driver performance, and manage maintenance logs. **[40 minutes, 4+4+5+4+4+4=25 marks]**

A. Create **TelemetryData Class** simulated telemetry data for vehicles , including attributes : The amount of fuel consumed by the vehicle, Fuel Consumption (liters per hour),The current speed of the vehicle Speed (kilometers per hour).The status of the vehicle's engine Engine Status(true if running, false otherwise).

B. Design a **Vehicle Class** to store the vehicle details, including attributes Vehicle ID ("V1234") ,Model ("Toyota Corolla 2022").Fuel Type ("Petrol" or "Diesel"), Current Location (“Karachi”) and Instance of Telemetry Data class
Implement the functions as described below

- displayDetails(): Displays detailed information about the vehicle, including its ID, model, fuel type, current location, and telemetry data .
- updateLocation(const string, newLocation): Updates the current location of the vehicle with the provided new location.
- getVehicleID() const: Retrieves the unique identifier of the vehicle.

C. Create a friend class of vehicle as **VehicleManager Class**: Responsible for tracking the location of vehicles using trackVehicleLocation(Vehicle vehicle): Tracks the location of a vehicle and displays the current location of the vehicle with its ID.
   i. Create function WriteToFile(vehicle vehiclesArray): writes the details of all vehicles provided to a file named "vehicle_info ".
   b. Create function ReadFromFile(): Read those vehicle from file whose location is **Karachi**.

D. **MeterReadingVehicle Class** Subclass of Vehicle, representing a vehicle used specifically for meter reading include attributes Reading Capacity(int), Overrides displayDetails() to display reading capacity.

E. **PipelineInspectionVehicle Class**:Subclass of Vehicle, representing a vehicle used for pipeline inspection include attribute InspectionRange(int), Overrides displayDetails() to dispay inspection range.

F. **MaintenanceVehicle Class**: Subclass of Vehicle Represents a vehicle used for maintenance purposes, include attribute Equipment Capacity (int): Capacity of the vehicle to carry maintenance equipment, measured in units. Overrides displayDetails() to include equipment capacity.

```
import java.io.*;
import java.util.*;

class TelemetryData {
   private float fuelConsumption; // liters per hour
   private float speed; // kilometers per hour
   private boolean engineStatus; // true if running, false otherwise

   public TelemetryData(float fuelConsumption, float speed, boolean engineStatus) {
      this.fuelConsumption = fuelConsumption;
      this.speed = speed;
      this.engineStatus = engineStatus;
   }

   public void displayTelemetry() {
      System.out.println("Telemetry Data:");
      System.out.println("Fuel Consumption: " + fuelConsumption + " liters/hour");
      System.out.println("Speed: " + speed + " km/h");
      System.out.println("Engine Status: " + (engineStatus ? "Running" : "Stopped"));
   }
}

class Vehicle implements Serializable {
   protected String vehicleID;
   protected String model;
   protected String fuelType;
   protected String currentLocation;
   protected TelemetryData telemetry;
```

```
  public Vehicle(String vehicleID, String model, String fuelType, String currentLocation, TelemetryData telemetry) {
      this.vehicleID = vehicleID;
      this.model = model;
      this.fuelType = fuelType;
      this.currentLocation = currentLocation;
      this.telemetry = telemetry;
  }

  public void displayDetails() {
      System.out.println("Vehicle ID: " + vehicleID);
      System.out.println("Model: " + model);
      System.out.println("Fuel Type: " + fuelType);
      System.out.println("Current Location: " + currentLocation);
      telemetry.displayTelemetry();
  }

  public void updateLocation(String newLocation) {
      this.currentLocation = newLocation;
  }
}

class VehicleManager {
  public void trackVehicleLocation(Vehicle vehicle) {
      System.out.println("Tracking Vehicle ID: " + vehicle.vehicleID + ", Current Location: " + vehicle.currentLocation);
  }

  public void writeToFile(Vehicle[] vehiclesArray) {
      try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("vehicle_info.txt"))) {
          for (Vehicle vehicle : vehiclesArray) {
              oos.writeObject(vehicle);
          }
      } catch (IOException e) {
          e.printStackTrace();
      }
  }

  public void readFromFile() {
      try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("vehicle_info.txt"))) {
          while (true) {
              try {
                  Vehicle vehicle = (Vehicle) ois.readObject();
                  if (vehicle.currentLocation.equals("Karachi")) {
                      System.out.println("Vehicle ID: " + vehicle.vehicleID + ", Model: " + vehicle.model + ", Fuel Type: " + vehicle.fuelType + ", Location: " + vehicle.currentLocation);
```

```
                }
            } catch (EOFException eof) {
                break;
            }
        }
    } catch (IOException) {
        e.printStackTrace();
    }
  }
}

class MeterReadingVehicle extends Vehicle implements Serializable {
  private int readingCapacity;

  public MeterReadingVehicle(String vehicleID, String model, String fuelType, String currentLocation,
TelemetryData telemetry, int readingCapacity) {
    super(vehicleID, model, fuelType, currentLocation, telemetry);
    this.readingCapacity = readingCapacity;
  }

  @Override
  public void displayDetails() {
    super.displayDetails();
    System.out.println("Reading Capacity: " + readingCapacity);
  }
}

class PipelineInspectionVehicle extends Vehicle implements Serializable {
  private int inspectionRange;

  public PipelineInspectionVehicle(String vehicleID, String model, String fuelType, String currentLocation,
TelemetryData telemetry, int inspectionRange) {
    super(vehicleID, model, fuelType, currentLocation, telemetry);
    this.inspectionRange = inspectionRange;
  }

  @Override
  public void displayDetails() {
    super.displayDetails();
    System.out.println("Inspection Range: " + inspectionRange + " km");
  }
}

class MaintenanceVehicle extends Vehicle implements Serializable {
  private int equipmentCapacity;
```

```
  public MaintenanceVehicle(String vehicleID, String model, String fuelType, String currentLocation,
TelemetryData telemetry, int equipmentCapacity) {
    super(vehicleID, model, fuelType, currentLocation, telemetry);
    this.equipmentCapacity = equipmentCapacity;
  }

  @Override
  public void displayDetails() {
    super.displayDetails();
    System.out.println("Equipment Capacity: " + equipmentCapacity + " units");
  }
}
```

CLO #4: Design and assess small and medium scale C++ / C# programs using object-oriented programming principles

---

Q5: Imagine you have been asked to develop a system for managing the Jewelry shop using Object-Oriented Programming (OOP) concepts. The system should handle various types of jewelry items, customer transactions, and inventory management. Consider the following detailed requirements for the system:

[45 minutes, 6*4+6=30 marks]

1. JewelryItem Class (Abstract):

- Create an abstract class called JewelryItem.
- Attributes: itemCode (string), itemName (string), weightInGrams (double), purity (int).
- Methods:
  - void displayDetails(): Display the details of the jewelry item including item code, item name, weight, and purity.
  - double calculatePrice(): Calculate and return the price of the jewelry item based on weight and purity.

2. GoldJewelry Class (Derived from JewelryItem):

- Inherits from the JewelryItem class.
- Additional Attribute: goldKarat (int).
- Methods:
  - void setGoldKarat(int karat): Set the gold karat value for the item.
  - Override calculatePrice() to calculate the price based on weight, purity, and gold karat.

3. DiamondJewelry Class (Derived from JewelryItem):

- Inherits from the JewelryItem class.
- Additional Attribute: numDiamonds (int), diamondCarat (double).
- Methods:
  - void addDiamonds(int num, double carat): Add diamonds to the jewelry item with the specified carat weight.
  - Override calculatePrice() to calculate the price based on weight, purity, and diamond carat weight.

4. Customer Class:

- Create a class called Customer to represent customers visiting the jewelry store.
- Attributes: customerID (string), name (string), purchasedItems (dynamic array to JewelryItem objects).

- Methods:
  - void purchaseItem(JewelryItem item): Add a purchased item to the customer's list of purchased items.
  - double calculateTotalPurchasePrice(): Calculate the total price of all purchased items.

5. StoreInventory Class:

- Create a class called StoreInventory to manage the store's inventory of jewelry items.
- Attributes: inventory (dynamic array to JewelryItem objects).
- Methods:
  - void addItemToInventory(JewelryItem item): Add a jewelry item to the store's inventory.
  - void displayInventory(): Display the details of all items in the inventory.

6. Draw UML/Class Diagram of the requirements given above.

```
import java.util.*;

// Abstract class JewelryItem
abstract class JewelryItem {
   protected String itemCode;
   protected String itemName;
   protected double weightInGrams;
   protected int purity;

   public JewelryItem(String code, String name, double weight, int pur) {
      itemCode = code;
      itemName = name;
      weightInGrams = weight;
      purity = pur;
   }

   abstract void displayDetails();
   abstract double calculatePrice();
}

// GoldJewelry class derived from JewelryItem
class GoldJewelry extends JewelryItem {
   private int goldKarat;

   public GoldJewelry(String code, String name, double weight, int pur) {
      super(code, name, weight, pur);
   }

   public void setGoldKarat(int karat) {
      goldKarat = karat;
   }

   @Override
   public void displayDetails() {
      System.out.println("Item Code: " + itemCode);
```

```
        System.out.println("Item Name: " + itemName);
        System.out.println("Weight: " + weightInGrams + " grams");
        System.out.println("Purity: " + purity + "%");
        System.out.println("Gold Karat: " + goldKarat);
    }

    @Override
    public double calculatePrice() {
        return weightInGrams * (purity / 100.0) * goldKarat * 50; // Example pricing formula
    }
}

// DiamondJewelry class derived from JewelryItem
class DiamondJewelry extends JewelryItem {
    private int numDiamonds;
    private double diamondCarat;

    public DiamondJewelry(String code, String name, double weight, int pur) {
        super(code, name, weight, pur);
    }

    public void addDiamonds(int num, double carat) {
        numDiamonds = num;
        diamondCarat = carat;
    }

    @Override
    public void displayDetails() {
        System.out.println("Item Code: " + itemCode);
        System.out.println("Item Name: " + itemName);
        System.out.println("Weight: " + weightInGrams + " grams");
        System.out.println("Purity: " + purity + "%");
        System.out.println("Number of Diamonds: " + numDiamonds);
        System.out.println("Diamond Carat: " + diamondCarat);
    }

    @Override
    public double calculatePrice() {
        return weightInGrams * (purity / 100.0) * diamondCarat * 1000; // Example pricing formula
    }
}

// Customer class
class Customer {
    private String customerID;
    private String name;
    private JewelryItem purchasedGold;
    private JewelryItem purchasedDiamond;

    public Customer(String id, String nm) {
```

```
        customerID = id;
        name = nm;
    }

    public void purchaseGold(GoldJewelry gold) {
        purchasedGold = gold;
    }

    public void purchaseDiamond(DiamondJewelry diamond) {
        purchasedDiamond = diamond;
    }

    public double calculateTotalPurchasePrice() {
        double total = 0.0;
        if (purchasedGold != null) {
            total += purchasedGold.calculatePrice();
        }
        if (purchasedDiamond != null) {
            total += purchasedDiamond.calculatePrice();
        }
        return total;
    }
}

// StoreInventory class
class StoreInventory {
    private List<JewelryItem> inventory;

    public StoreInventory() {
        inventory = new ArrayList<>();
    }

    public void addItemToInventory(JewelryItem item) {
        inventory.add(item);
    }

    public void displayInventory() {
        if (inventory.isEmpty()) {
            System.out.println("Inventory is empty.");
        } else {
            System.out.println("Store Inventory:");
            for (JewelryItem item : inventory) {
                System.out.println("Item Code: " + item.itemCode);
                System.out.println("Item Name: " + item.itemName);
                System.out.println("Weight: " + item.weightInGrams + " grams");
                System.out.println("Purity: " + item.purity + "%");
                System.out.println("Price: $" + item.calculatePrice());
                System.out.println();
            }
        }
```

```
  }
}
```

CLO #5: Synthesize programs using Generic Programming and exception handling

Q6: Imagine you are developing a shopping cart system for an e-commerce website. The shopping cart should be able to hold up to 10 unique items of different product types that are book, electronics & clothing. Create and implement a template class named UniqueCart to manage the shopping cart's functionalities for various product types. **[30 minutes, 20 marks]**

A. **Add:** Implement a method to add items to the cart (void add(const T item)), where T represents three different product types that are books, electronics & clothing.
B. **Remove:** Implement a method to remove items from the cart (void remove(const T item)).
C. **Contains:** Implement a method to check if a specific item exists in the cart (bool contains(const Titem)). Your implementation should also handle the exceptions for the following scenarios:
    a. Throw an exception "Duplicate Item Exception" when trying to add a duplicate item to the cart.
    b. Throw an exception "Item Not Found Exception" when trying to remove an item that does not exist in the cart.
    c. Throw an exception "Out of bound" when the cart capacity (10 items) is exceeded.

```
import java.util.*;

// Custom exception classes
class DuplicateItemException extends Exception {
  public String find() {
    return "Duplicate Item Exception";
  }
}

class ItemNotFoundException extends Exception {
  public String find() {
    return "Item Not Found Exception";
  }
}

class OutOfBoundException extends Exception {
  public String find() {
    return "Out of bound";
  }
}
```

```
// Base class for products
abstract class Product {
    abstract void print();
    abstract boolean equals(Product other);
}

// Specific product types
class Book extends Product {
    String title;
    String author;

    public Book(String bookTitle, String bookAuthor) {
        title = bookTitle;
        author = bookAuthor;
    }

    public void print() {
        System.out.println("Book: " + title + " by " + author);
    }

    public boolean equals(Product other) {
        if (other instanceof Book) {
            Book otherBook = (Book) other;
            return title.equals(otherBook.title) && author.equals(otherBook.author);
        }
        return false;
    }
}

class Electronic extends Product {
    String name;
    double price;

    public Electronic(String itemName, double itemPrice) {
        name = itemName;
        price = itemPrice;
    }

    public void print() {
        System.out.println("Electronic: " + name + " costs $" + price);
    }
```

```
    public boolean equals(Product other) {
        if (other instanceof Electronic) {
            Electronic otherElectronic = (Electronic) other;
            return name.equals(otherElectronic.name) && price == otherElectronic.price;
        }
        return false;
    }
}

class Clothing extends Product {
    String type;
    String size;

    public Clothing(String itemType, String itemSize) {
        type = itemType;
        size = itemSize;
    }

    public void print() {
        System.out.println("Clothing: " + type + " size " + size);
    }

    public boolean equals(Product other) {
        if (other instanceof Clothing) {
            Clothing otherClothing = (Clothing) other;
            return type.equals(otherClothing.type) && size.equals(otherClothing.size);
        }
        return false;
    }
}

// Define the UniqueCart template class
class UniqueCart<T extends Product> {
    T[] items;
    int itemCount;

    public UniqueCart() {
        items = (T[]) new Product[10];
        itemCount = 0;
    }
```

```
// Method to add items to the cart
public void add(T item) throws OutOfBoundException, DuplicateItemException {
    if (itemCount >= 10) {
        throw new OutOfBoundException();
    }

    for (int i = 0; i < itemCount; ++i) {
        if (items[i] != null && items[i].equals(item)) {
            throw new DuplicateItemException();
        }
    }

    items[itemCount++] = item;
    System.out.println("Item added to cart.");
}

// Method to remove items from the cart
public void remove(T item) throws ItemNotFoundException {
    boolean found = false;
    for (int i = 0; i < itemCount; ++i) {
        if (items[i] != null && items[i].equals(item)) {
            found = true;
            System.out.println("Item removed from cart.");
            items[i] = null;
            --itemCount;
            // Shift items to fill the gap
            for (int j = i; j < itemCount; ++j) {
                items[j] = items[j + 1];
            }
            items[itemCount] = null;
            break;
        }
    }
    if (!found) {
        throw new ItemNotFoundException();
    }
}

// Method to check if a specific item exists in the cart
public boolean contains(T item) {
    for (int i = 0; i < itemCount; ++i) {
        if (items[i] != null && items[i].equals(item)) {
```

```
            return true;
        }
      }
      return false;
    }

    // Method to print all items in the cart
    public void print() {
      for (int i = 0; i < itemCount; ++i) {
        items[i].print();
      }
    }
}

public class Main {
    public static void main(String[] args) {
      // Create a cart that can hold up to 10 items of various types
      UniqueCart<Product> cart = new UniqueCart<>();

      // Add items to the cart
      try {
        cart.add(new Book("The Great Gatsby", "F. Scott Fitzgerald"));
        cart.add(new Electronic("Laptop", 1200.0));
        cart.add(new Clothing("T-shirt", "Medium"));
        // Adding a duplicate item
        // cart.add(new Book("The Great Gatsby", "F. Scott Fitzgerald"));
      } catch (OutOfBoundException | DuplicateItemException ex) {
        System.out.println("Error: " + ex.find());
      }

      // Print all items in the cart
      cart.print();

      // Create items to remove/check
      Book book = new Book("The Great Gatsby", "F. Scott Fitzgerald");
      Electronic laptop = new Electronic("Laptop", 1200.0);

      // Remove an item from the cart
      try {
        cart.remove(book);
        // Trying to remove a non-existing item
        // cart.remove(laptop);
```

```
    } catch (ItemNotFoundException ex) {
       System.out.println("Error: " + ex.find());
    }

    // Check if items are in the cart
    System.out.println("Cart contains The Great Gatsby: " + cart.contains(book));
    System.out.println("Cart contains Laptop: " + cart
}
```