What is the purpose of generalization in object-oriented modeling?  
  
**1: 2 ||| 11% 2: 13 ||||||||||||||| 72% 3: 2 ||| 11% 4: 1 || 6%   
  
Polling options**1: To create objects from classes 2: To define a relationship bw classes & subclas 3: To encapsulate data within objects 4: To handle object persistence

A: 2

Interfaces in object-oriented modeling define:  
  
**1: 1 || 5% 2: 3 ||| 14% 3: 16 |||||||||||||||| 76% 4: 1 || 5%   
  
Polling options**1: The hierarchy of classes and subclasses 2: The relationship between classes and subclass 3: A set of methods that a class must implement 4: The visibility of features within a class

A: 3

**Exercise:**

Student - class, (name, age, major) variables

import java.io.\*;

class Student implements Serializable {

    private String name;

    private int age;

    private String major;

    public Student(String name, int age, String major) {

        this.name = name;

        this.age = age;

        this.major = major;

    }

    public String getName() {

        return name;

    }

    public int getAge() {

        return age;

    }

    public String getMajor() {

        return major;

    }

}

public class Main {

    public static void main(String args[]){

        try {

            Student student = new Student("Saswath",28,"CSE");

            FileOutputStream fileOut = new FileOutputStream("student.ser");

            ObjectOutputStream out = new ObjectOutputStream(fileOut);

            out.writeObject(student);

            out.close();

            fileOut.close();

            System.out.println("Wrote student object to file system");

        }

        catch (IOException e) {

            System.out.println(e);

        }

        try {

            FileInputStream input = new FileInputStream("student.ser");

            ObjectInputStream obin = new ObjectInputStream(input);

            Student student = (Student) obin.readObject();

            input.close();

            obin.close();

            System.out.println("Read Serialized Object");

            System.out.println(student.getName());

            System.out.println(student.getAge());

            System.out.println(student.getMajor());

        }

        catch (IOException | ClassNotFoundException e){

            System.out.println(e);

        }

    }

}

Persistence save(), load()

import java.io.\*;

class Persistent implements Serializable {

    String objName;

    Persistent(String objName){

        this.objName = objName;

    }

    public void save(){

        try {

            FileOutputStream fout = new FileOutputStream(this.objName+".ser");

            ObjectOutputStream objout = new ObjectOutputStream(fout);

            objout.writeObject(this);

            objout.close();

            fout.close();

            System.out.println(this.objName+".ser is saved to file system");

        }

        catch (IOException e){

            System.out.println(e);

        }

    }

    public Persistent load(String objName){

        try{

            FileInputStream fin = new FileInputStream(objName+".ser");

            ObjectInputStream oin = new ObjectInputStream(fin);

            Persistent p = (Persistent) oin.readObject();

            fin.close();

            oin.close();

            System.out.println(objName+".ser successfully read by program");

            return p;

        }

        catch (IOException | ClassNotFoundException e){

            System.out.println(e);

            return this;

        }

    }

}

class Student extends Persistent {

    private String name;

    private int age;

    private String major;

    public Student(String name, int age, String major) {

        super(name);

        this.name = name;

        this.age = age;

        this.major = major;

    }

    public Student(){

        super("");

        this.name = "";

        this.age = 0;

        this.major = "";

    }

    public String getName() {

        return name;

    }

    public int getAge() {

        return age;

    }

    public String getMajor() {

        return major;

    }

}

class Employee extends Persistent {

    String name;

    Employee(String name){

        super(name);

        this.name = name;

    }

    Employee(){

        super("");

        this.name = "";

    }

}

public class Main {

    public static void main(String args[]){

        Student s = new Student("Saswath",28,"CSE");

        s.save();

        Student s2 = new Student();

        s2 = (Student) s2.load("Saswath");

        System.out.println(s2.getAge());

        System.out.println(s2.getMajor());

        Employee e = new Employee("Krishna");

        e.save();

        Employee e2 = new Employee();

        e2 = (Employee) e2.load("Krishna");

        System.out.println(e2.name);

    }

}

Q1)

public class BankAccount {

private String accountNumber;

private String accountHolder;

private double balance;

private boolean isSavingsAccount;

public BankAccount(String accountNumber, String accountHolder, double balance, boolean isSavingsAccount) {

this.accountNumber = accountNumber;

this.accountHolder = accountHolder;

this.balance = balance;

this.isSavingsAccount = isSavingsAccount;

}

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

} else {

System.out.println("Insufficient funds.");

}

}

public void transfer(BankAccount destinationAccount, double amount) {

if (amount <= balance) {

balance -= amount;

destinationAccount.deposit(amount);

} else {

System.out.println("Insufficient funds.");

}

}

// Getters and setters for all fields

}

Q2)

import java.util.ArrayList;

import java.util.List;

public class Library {

private List<Book> books;

private List<User> users;

private List<BorrowedBook> borrowedBooks;

public Library() {

this.books = new ArrayList<>();

this.users = new ArrayList<>();

this.borrowedBooks = new ArrayList<>();

}

public void addBook(Book book) {

books.add(book);

}

public void removeBook(Book book) {

books.remove(book);

}

public void addUser(User user) {

users.add(user);

}

public void removeUser(User user) {

users.remove(user);

}

public boolean borrowBook(Book book, User user) {

if (books.contains(book) && users.contains(user)) {

BorrowedBook borrowedBook = new BorrowedBook(book, user);

borrowedBooks.add(borrowedBook);

return true;

} else {

return false;

}

}

public boolean returnBook(Book book, User user) {

BorrowedBook borrowedBook = new BorrowedBook(book, user);

if (borrowedBooks.contains(borrowedBook)) {

borrowedBooks.remove(borrowedBook);

return true;

} else {

return false;

}

}

// Getters and setters for books, users, and borrowedBooks

}

Hint for question 2:

class Book{

    String name;

    String author;

    Book(String name, String author){

        this.name = name;

        this.author = author;

    }

}

class Test{

    boolean addTest(){

        Library library = new Library();

        Book book1 = new Book("HarryPotter1", "J.K.Rowling");

        Book book2 = new Book("HarryPotter2", "J.K.Rowling");

        Book book3 = new Book("HarryPotter3", "J.K.Rowling");

        library.addBook(book1);

        library.addBook(book2);

        library.addBook(book3);

        if(library.getCount() == 3){

            return true;

        }

        else{

            return false;

        }

    }

}

Q3)

public class Customer {

private String customerId;

private String customerName;

private String customerType;

private double discountPercentage;

private double totalPurchaseAmount;

private boolean isLoyalCustomer;

public Customer(String customerId, String customerName, String customerType, double discountPercentage,

double totalPurchaseAmount, boolean isLoyalCustomer) {

this.customerId = customerId;

this.customerName = customerName;

this.customerType = customerType;

this.discountPercentage = discountPercentage;

this.totalPurchaseAmount = totalPurchaseAmount;

this.isLoyalCustomer = isLoyalCustomer;

}

public void calculateDiscountedAmount() {

double discountedAmount = totalPurchaseAmount - (totalPurchaseAmount \* discountPercentage);

System.out.println("Discounted Amount: " + discountedAmount);

}

public void displayCustomerDetails() {

System.out.println("Customer ID: " + customerId);

System.out.println("Customer Name: " + customerName);

System.out.println("Customer Type: " + customerType);

System.out.println("Total Purchase Amount: " + totalPurchaseAmount);

System.out.println("Is Loyal Customer: " + isLoyalCustomer);

}

public void printInvoice() {

System.out.println("Invoice Details:");

System.out.println("Customer ID: " + customerId);

System.out.println("Customer Name: " + customerName);

System.out.println("Customer Type: " + customerType);

System.out.println("Total Purchase Amount: " + totalPurchaseAmount);

System.out.println("Discount Percentage: " + discountPercentage);

System.out.println("Is Loyal Customer: " + isLoyalCustomer);

calculateDiscountedAmount();

}

public boolean isPremiumCustomer() {

return customerType.equals("Premium");

}

public boolean isLoyalPremiumCustomer() {

return customerType.equals("Premium") && isLoyalCustomer;

}

// Getters and setters for all fields

}