Java Assignment Answers

## 1. Use javadoc comment in a program to demonstrate it. Also explain javah and javap with example.

/\*\*  
 \* This class represents a simple HelloWorld example.  
 \* @author Saish  
 \* @version 1.0  
 \*/  
public class HelloWorld {  
 public static void main(String[] args) {  
 System.out.println("Hello, world!");  
 }  
}  
  
/\*  
javap Example:  
> javap HelloWorld  
  
javah (deprecated) was used to generate C header files from Java classes.  
Now replaced by 'javac -h'.  
\*/

## 2. Constructor Overloading with Employee class:

class Employee {  
 int ecode;  
 String ename, desg, mobile, email;  
 double salary;  
  
 Employee() {  
 this(0, "Unknown", 0.0, "None", "0000", "noemail@example.com");  
 }  
  
 Employee(int ecode, String ename) {  
 this(ecode, ename, 0.0, "Unknown", "0000", "noemail@example.com");  
 }  
  
 Employee(int ecode, String ename, double salary, String desg, String mobile, String email) {  
 this.ecode = ecode;  
 this.ename = ename;  
 this.salary = salary;  
 this.desg = desg;  
 this.mobile = mobile;  
 this.email = email;  
 }  
  
 void dispDetails() {  
 System.out.println("Code: " + ecode + ", Name: " + ename + ", Salary: " + salary +  
 ", Designation: " + desg + ", Mobile: " + mobile + ", Email: " + email);  
 }  
}

## 3. Person and Customer Class with super():

class Person {  
 String aadhar, name, address;  
  
 Person() {}  
 Person(String aadhar, String name) {  
 this.aadhar = aadhar;  
 this.name = name;  
 }  
 Person(String aadhar, String name, String address) {  
 this(aadhar, name);  
 this.address = address;  
 }  
  
 void show() {  
 System.out.println("Name: " + name + ", Aadhar: " + aadhar);  
 }  
}  
  
class Customer extends Person {  
 String cust\_code, username, password, mobile, email;  
  
 Customer(String aadhar, String name, String address, String cust\_code, String username, String password, String mobile, String email) {  
 super(aadhar, name, address);  
 this.cust\_code = cust\_code;  
 this.username = username;  
 this.password = password;  
 this.mobile = mobile;  
 this.email = email;  
 }  
  
 void display() {  
 super.show();  
 System.out.println("Customer Code: " + cust\_code + ", Email: " + email);  
 }  
}

## 4. Access Specifiers & Modifiers (9 examples):

Access Specifiers:  
1. public: accessible everywhere.  
2. private: accessible only within class.  
3. protected: accessible in same package or subclass.  
4. default (no keyword): accessible in same package.  
  
Modifiers:  
5. static: belongs to class not instance.  
6. final: cannot be modified (method, variable, class).  
7. abstract: declared but not implemented.  
8. synchronized: used in threading.  
9. transient: skip during serialization.

## 5. Dynamic Method Dispatch:

class Animal {  
 void sound() { System.out.println("Some sound"); }  
}  
  
class Dog extends Animal {  
 void sound() { System.out.println("Bark"); }  
}  
  
public class Test {  
 public static void main(String[] args) {  
 Animal a = new Dog(); // upcasting  
 a.sound(); // calls Dog's method at runtime  
 }  
}

## 6. Nested & Inner Classes:

class Outer {  
 int x = 10;  
  
 class Inner {  
 void show() {  
 System.out.println("Inner class: x = " + x);  
 }  
 }  
  
 static class StaticInner {  
 void display() {  
 System.out.println("Static inner class.");  
 }  
 }  
  
 void methodWithLocalClass() {  
 class Local {  
 void print() { System.out.println("Local class."); }  
 }  
 new Local().print();  
 }  
  
 void anonymousClass() {  
 Runnable r = new Runnable() {  
 public void run() { System.out.println("Anonymous class."); }  
 };  
 r.run();  
 }  
}

## 7. Abstract Product with Food and Drink:

abstract class Product {  
 String name;  
 double price;  
  
 Product(String name, double price) {  
 this.name = name;  
 this.price = price;  
 }  
  
 abstract void displayDetails();  
}  
  
class Food extends Product {  
 int calories;  
 Food(String name, double price, int calories) {  
 super(name, price);  
 this.calories = calories;  
 }  
 void displayDetails() {  
 System.out.println("Food: " + name + ", " + price + ", " + calories);  
 }  
}  
  
class Drink extends Product {  
 boolean isCold;  
 Drink(String name, double price, boolean isCold) {  
 super(name, price);  
 this.isCold = isCold;  
 }  
 void displayDetails() {  
 System.out.println("Drink: " + name + ", " + price + ", Cold: " + isCold);  
 }  
}

## 8. Factory Pattern using ProductManager:

class ProductManager {  
 public static Food createFood(String name, double price, int calories) {  
 return new Food(name, price, calories);  
 }  
 public static Drink createDrink(String name, double price, boolean isCold) {  
 return new Drink(name, price, isCold);  
 }  
}  
  
class Shop {  
 public static void main(String[] args) {  
 Product f = ProductManager.createFood("Pizza", 100, 300);  
 Product d = ProductManager.createDrink("Cola", 50, true);  
 f.displayDetails();  
 d.displayDetails();  
 }  
}

## 9. Rateable Interface & Reviews:

interface Rateable {  
 void addReview(Review r);  
 void printReviews();  
}  
  
class Review {  
 int rating;  
 String comment;  
 Review(int rating, String comment) {  
 this.rating = rating;  
 this.comment = comment;  
 }  
 public String toString() {  
 return rating + "★ - " + comment;  
 }  
}  
  
abstract class Product implements Rateable {  
 String name;  
 double price;  
 List<Review> reviews = new ArrayList<>();  
 public void addReview(Review r) { reviews.add(r); }  
 public void printReviews() {  
 for (Review r : reviews) System.out.println(r);  
 }  
}  
  
class ProductManager {  
 public static void addProductReview(Product p, int rating, String comment) {  
 p.addReview(new Review(rating, comment));  
 }  
}

## 10. sports\_accessories class for Cricket:

class sports\_accessories {  
 int id, quantity;  
 String desc, game;  
 double rate;  
  
 void acceptDetails(Scanner sc) {  
 // input fields  
 }  
 void displayDetails() {  
 // print fields  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 sports\_accessories[] arr = new sports\_accessories[5];  
 // input and filter cricket accessories  
 }  
}

## 11. Arrays class methods:

int[] arr = {5,2,8,1,9};  
Arrays.sort(arr);  
Arrays.binarySearch(arr, 8);  
Arrays.copyOf(arr, 5);  
Arrays.copyOfRange(arr, 1, 4);  
Arrays.equals(arr, arr2);  
Arrays.fill(arr, 7);  
Arrays.setAll(arr, i -> i \* 2);  
Arrays.mismatch(arr1, arr2);  
Arrays.stream(arr).filter(x -> x > 4).count();  
Arrays.toString(arr);

## 12. Student class with sorting by percentage:

class Student {  
 String name, className;  
 int roll\_no,[] marks = new int[6];  
 double percentage;  
  
 void per() { /\* calc percentage \*/ }  
 static void sortStudent(Student[] arr) {  
 Arrays.sort(arr, (s1, s2) -> Double.compare(s2.percentage, s1.percentage));  
 }  
}