Q1. Create a class Person with attributes such as name and age. Derive a class Student from Person with additional attributes like studentId and grade. Demonstrate inheritance by creating objects of both classes and accessing their attributes.

class Person {  
 String name;  
 int age;  
 Person()  
 {  
 name="Rupkanth";  
 age=23;  
 }  
  
}  
class Student extends Person{  
 int studentId;  
 char grade;  
 Student()  
 {  
 studentId=17;  
 grade='O';  
 }  
}  
class Person1 {  
 public static void main(String[] args)  
 {  
 Person p=new Person();  
 Student s=new Student();  
 System.*out*.println("Student Name "+p.name);  
 System.*out*.println("Student age "+p.age);  
 System.*out*.println("=============================");  
 System.*out*.println("Student Name "+s.name);  
 System.*out*.println("Student age "+s.age);  
 System.*out*.println("Student Id "+s.studentId);  
 System.*out*.println("Student grade "+s.grade);  
 }  
  
}

Q2. Implement a class hierarchy for geometric shapes, starting with a base class Shape. Derive classes such as Circle and Rectangle. Include methods to calculate the area of each shape. Demonstrate the use of these classes in a program.

class Shape {  
 int l, b;  
 double r;  
 Shape() {  
 l=5;  
 b=6;  
 r=5.5;  
 }  
  
 public void area() {  
 System.*out*.println("Shapes");  
 }  
}  
class Circle extends Shape{  
 final double PI=3.14;  
 public void area()  
 {  
 System.*out*.println("Area of circle "+(PI\*r\*r));  
 }  
}  
class Rectangle extends Shape  
{  
 public void area()  
 {  
 System.*out*.println("Area of rectangle "+(l\*b));  
 }  
}  
public class Shape1 {  
 public static void main(String[] args)  
 {  
 Shape s;  
 s=new Circle();  
 s.area();  
 s=new Rectangle();  
 s.area();  
 }  
}

Q3.Define a class Vehicle with a method start(). Derive classes Car and Motorcycle. Override the start method in each derived class to provide specific behavior. Create objects of both classes and call the start method to observe polymorphic behavior.

class Vehicle{  
 public void start(){  
 System.*out*.println("Start to move vehicle");  
 }  
}  
class Car extends Vehicle{  
 public void start()  
 {  
 System.*out*.println("car started");  
 }  
}  
class Motorcycle extends Vehicle{  
 public void start()  
 {  
 System.*out*.println("motorcycle started");  
 }  
}  
public class Vehicle1 {  
 public static void main(String[] args)  
 {  
 Vehicle c;  
 c=new Car();  
 c.start();  
 c=new Motorcycle();  
 c.start();  
 }  
}

Q4. Create a base class Employee with a method calculateSalary. Derive classes Manager and Worker. Override the calculateSalary method in each derived class to include bonuses for managers and overtime pay for workers.

class Employee  
{  
 public void calculateSalary(int salary,int bonus,float over)  
 {  
 float a=salary+bonus+over;  
 System.*out*.println("Total salary of employee "+a);  
 }  
}  
class Manager extends Employee  
{  
 String name="Rupkanth";  
 public void calculateSalary(int salary,int bonus,float over)  
 {  
 float a=salary+bonus+over;  
 System.*out*.println("Employee name "+name);  
 System.*out*.println("Salary of manager "+a);  
 }  
}  
class Worker extends Employee  
{  
 String name="Vamsi Krishna";  
 public void calculateSalary(int salary,int bonus,float over)  
 {  
 float a=salary+bonus+over;  
 System.*out*.println("Employee name "+name);  
 System.*out*.println("Salary of manager "+a);  
 }  
  
}  
public class Employee1 {  
 public static void main(String[] args){  
 Employee e;  
 e=new Manager();  
 e.calculateSalary(50000,3000,2000);  
 System.*out*.println("===============================");  
 e=new Worker();  
 e.calculateSalary(30000,2000,1000);  
 }  
}

Q5.Create a base class Person with a parameterized constructor. Derive a class Employee with additional attributes. Demonstrate how the constructor of the base class can be called from the derived class using the super keyword.

class Pers{  
 String name="Rupkanth",location="Guntur";  
 int age=23;  
 /\*  
 Pers(String name,int age,String location)  
 {  
 this.name=name;  
 this.age=age;  
 this.location=location;  
 }  
 \*/  
 public void print()  
 {  
 System.*out*.println("Name "+name);  
 System.*out*.println("Age "+age);  
 System.*out*.println("Location "+location);  
 }  
}  
class Emp extends Pers  
{  
 int employeeId;  
 String post;  
 Emp(int employeeId,String post)  
 // Emp(String name,int age,String location,int employeeId,String post)  
 {  
 //super(name,age,location);  
 this.employeeId=employeeId;  
 this.post=post;  
 }  
  
 public void print()  
 {  
 super.print();  
 System.*out*.println("Employee Id "+employeeId);  
 System.*out*.println("Designation "+post);  
 }  
}  
public class Person2 {  
 public static void main(String[] args) {  
 // Emp e = new Emp("Rupkanth", 23, "Guntur",1728,"M.D");  
 Emp e = new Emp(1728,"M.D");  
 e.print();  
 }  
}

Q6. Implement a class hierarchy for a zoo. Create a base class Animal with attributes like name and age. Derive classes such as Lion and Elephant. Include constructors in both the base and derived classes.

class Zoo  
{  
 String name;  
 int age;  
 Zoo(String name,int age)  
 {  
 this.name=name;  
 this.age=age;  
 }  
 public void print()  
 {  
 System.*out*.println("Animal name "+name);  
 System.*out*.println("Animal age "+age);  
 }  
}  
class Lion extends Zoo  
{  
 String type;  
 Lion(String name,int age,String type)  
 {  
 super(name,age);  
 this.type=type;  
 }  
 public void print() {  
 super.print();  
 System.*out*.println("Animal type "+type);  
 }  
}  
class Elephant extends Zoo  
{  
 String type;  
 Elephant(String name,int age,String type)  
 {  
 super(name,age);  
 this.type=type;  
 }  
 public void print() {  
 super.print();  
 System.*out*.println("Animal type "+type);  
 }  
}  
public class Zoo1 {  
 public static void main(String[] args){  
 Zoo z;  
 z=new Lion("Lion",10,"Carnivore");  
 z.print();  
 System.*out*.println("==============================");  
 z=new Elephant("Elephant",15,"Herbivore");  
 z.print();  
 }  
}

Q7. Create an interface Swim with a method swimDistance. Implement this interface in a class Fish. Also, create a class Bird with a method flyHeight. Derive a class Penguin from both Fish and Bird. Demonstrate how multiple inheritance is achieved through interfaces.

interface Swim {  
 void swimDistance();  
}  
interface Fly {  
 void flyHeight();  
}  
class Fish implements Swim {  
 public void swimDistance() {  
 System.*out*.println("Fish swims a long distance.");  
 }  
}  
class Bird implements Fly {  
 public void flyHeight() {  
 System.*out*.println("Bird flies at a high altitude.");  
 }  
}  
class Penguin extends Fish implements Fly {  
 public void flyHeight() {  
 System.*out*.println("Penguin cannot fly but can swim.");  
 }  
}  
public class Fish1 {  
 public static void main(String[] args) {  
 Fish fish = new Fish();  
 Bird bird = new Bird();  
 fish.swimDistance();  
 bird.flyHeight();  
 Penguin penguin = new Penguin();  
 penguin.swimDistance();  
 penguin.flyHeight();  
 }  
  
  
}

Q8. Discuss the concept of multiple inheritance in Java. Provide an example of a situation where multiple inheritance is beneficial and how it can be implemented using interfaces.

interface Walkable {  
 void walk();  
}  
interface Swimmable {  
 void swim();  
}  
class Duck implements Walkable, Swimmable {  
 public void walk()  
 {  
 System.*out*.println("Duck is walking.");  
 }  
 public void swim()  
 {  
 System.*out*.println("Duck is swimming.");  
 }  
}  
public class CS {  
  
 public static void main(String[] args) {  
 Duck duck = new Duck();  
 duck.walk();  
 duck.swim();  
 }  
}