Question 1 (compile time polymorphism)
Create a class MathOperations with overloaded methods(compile-time polymorphism) add. The add method should:

- Accept two integers and return their sum.
- Accept two double values and return their sum.
- Accept three integers and return their sum.

Here is the main method for this question:

```
public class Test {
   public static void main(String[] args) {
      MathOperations math = new MathOperations();
      System.out.println(math.add(2, 3));         // Output: 5
      System.out.println(math.add(2.5, 3.5));      // Output: 6.0
      System.out.println(math.add(1, 2, 3));       // Output: 6
   }
}
```

Question 2 (runtime polymorphism)
Create classes Animal, and Dog. Demonstrate runtime polymorphism(dynamic polymorphism) through method overriding. Animal class function makeSound prints “default animal sound”, Dog class function makeSound prints “hav hav”. In the main method upcast dog instance. Then, override with the makeSound function. Here is the main method for this question:

```
public class Main {
   public static void main(String[] args) {
      Animal animal = new Dog(); // Upcasting
      animal.makeSound(); // Output: Dog barks
   }
}
```

Question 3 (upcasting, downcasting)
Assume that we have classes, Employee, SoftwareDeveloper and BusinessAnalyst. Employee class is the superclass of SoftwareDeveloper and BusinessAnalyist. In the main method, demonstrate Upcasting(implicit casting) with SoftwareDeveloper and Employee objects. Then demonstrate downcasting(explicit casting) with BusinessAnalyst and Employee objects. Assume that constructors do not require parameters.

Question 4 (instanceof)
SoftwareDeveloper has a function called code(). BusinessAnalyst has a function called document(). Upcast two employee objects; one SoftwareDeveloper, and one BusinessAnalyist. Add these to an Employee arraylist. For each employee object in the list, call the respective function based on the instance type.

Question 5 (multilevel inheritance)
Create a base class Animal, a subclass Mammal, and a subclass Dog that extends Mammal. Write methods in each class that demonstrate multilevel inheritance. Animal class has a method eat, Mammal class has a method called breathe, Dog class has a method called bark.Create a dog instance and call every method from a main method.

Question 6 and 7 are followup questions
Question 6 (overriding)
Create a superclass Shape with a method area() which prints “area calculations goes here”. Create a subclass called Circle that overrides the area method, calculate the area of the circle and print it.

Question 7 (overriding)
Create a subclass called Rectangle that overrides the area method, calculate the area of the circle and print it. Also write the main function, create one circle and one rectangle instance. Then call area methods for each instance.

Question 8 (ArrayLists, instanceof)
In main method create an ArrayList that can accept any object. Then add one integer 1, string "1" and double 1.0 to the list. Then print the datatype and value for each element.

Question 9 (Method Overloading)
Create a class StringOperations with overloaded methods concat. The concat method should:

- Accept two strings and return their concatenation.
- Accept three strings and return their concatenation.
- Accept an array of strings and return their concatenation.

Question 10(Method Overriding)
Create a class Vehicle with a method move() that prints "Vehicle is moving." Create two subclasses Car and Bicycle that override the move() method with specific messages. In the main method, upcast Car and Bicycle objects and call their respective move() methods.

Question 11(Multilevel Inheritance)
Create a base class Plant, a subclass FloweringPlant, and a subclass Rose that extends FloweringPlant. Write methods in each class to demonstrate multilevel inheritance. The Plant class has a method grow(), FloweringPlant has a method bloom(), and Rose has a method smell(). These methods just prints some text. Call every method with a rose object. Here is the main method for this question:

```
public class Test {
    public static void main(String[] args) {
        Rose rose = new Rose();
        rose.grow();
        rose.bloom();
        rose.smell();
    }
}
```

Question 12 and 13 is follow-up questions

Question 12 (overriding)

Create a class Solid with a method volume() that prints a message saying, "Volume calculation for solid." Create subclass Prism that override the volume() method. In the subclass, calculate the volume of the rectangular prism.

Question 13 (overriding)

Create subclass Sphere that override the volume() method. In the subclass, calculate the volume of the sphere. In the main method create instances for the both subclasses and calculate volumes. Here is the main method for this question:

```
public class Test {
   public static void main(String[] args) {
      Solid sphere = new Sphere(3);
      Solid prism = new Prism(10, 5);

      sphere.volume();
      prism.volume();
   }
}
```

Question 14 (implicitly final methods)

Create a class Appliance with a method turnOn() that prints "Appliance is turning on." Make this method implicitly final in two ways.

Question 15 (super, overriding)

Create a class Employee with a method work() that prints "Employee is working." Create a subclass Manager that overrides the work() method to print "Manager is managing." In the Manager class, use the super keyword to call the work() method from the Employee class and demonstrate how it's used in the Manager class. Here is the main method for this question:

```
public class Test {
   public static void main(String[] args) {
      Manager manager = new Manager();
      manager.work();
   }
}
```

Question 16 (super, overriding, upcasting)
Create a class Shape with a method draw() that prints "Drawing shape." Create a subclass Rectangle that overrides draw() to print "Drawing rectangle." In the main method create a upcasted rectangle object. Modify the code with super keyword that allows us to print both "Drawing shape." and "Drawing rectangle." lines back to back with a single draw method call. Here is the main method for this question:

```
public class Test {
   public static void main(String[] args) {
      Shape shape = new Rectangle();
      shape.draw();
   }
}
```

Question 17 (super constructor)
Create a base class Person with a constructor that initializes the name. Create a subclass Student that has a constructor which calls the Person constructor using super() to initialize the name, and then initializes the student ID.

Question 18 (overloading, ArrayList)
Create a class ListManager with overloaded methods for adding elements to an ArrayList. The class should have the following methods:
- addElement(ArrayList<String> list, String element) - Adds a single String element to the ArrayList.
- addElement(ArrayList<String> list, List<String> elements) - Adds multiple String elements to the ArrayList.
- addElement(ArrayList<Integer> list, Integer element) - Adds a single Integer element to the ArrayList.
- addElement(ArrayList<Integer> list, List<Integer> elements) - Adds multiple Integer elements to the ArrayList.

Implement the ListManager class and demonstrate the functionality of these overloaded methods in the main method by creating ArrayList objects for String and Integer types, and adding elements using each method.

Question 19 and 20 are followup questions
Question 19 (Overriding)
Create a base class Sequence with a method generate(int n) that prints a sequence of numbers. Create a subclass PowerOfTwoSequence that overrides generate(int n) to print the first n powers of 2 (2^0, 2^1, 2^2, ...).

Question 20 (Overriding)
Create a subclass PascalTriangle that overrides generate(int n) to print the first n rows of Pascal's Triangle.