1.Human

2.Student

3.Professor

4.Main The professor and student classes inherit from the human class.

Question 1: In the Human class, change the staticPrint() method so that its output cannot be changed in the classes that implement the Human class.

Answer: To make the staticPrint() method in the Human class unable to be changed by its subclasses, you can make it final. This will prevent any subclasses from overriding the method. Here's how you can modify the Human class:

```
public class Human {
    public final void staticPrint() {
        System.out.println("This is a static print from Human class.");
    }
}
```

Now, the ‘staticPrint()’ method cannot be overridden by the ‘Student’ or ‘Professor’ classes.

Question 2: Modify the Student class so that when the sayMyName() method is called on it, its fullName attribute is printed in the output.

Answer:To modify the Student class so that the ‘sayMyName()’ method prints the ‘fullName’ attribute, you can simply implement the ‘sayMyName()’ method in the Student class to print the ‘fullName’. Here's how you can do it:

```
public class Student extends Human {
    private String fullName;

    public Student(String fullName) {
```

```
        this.fullName = fullName;
    }

    public void sayMyName() {
        System.out.println("My name is " + fullName);
    }
}
```

Now, when you call the ‘sayMyName()’ method on a ‘Student’ object, it will print the student's full name.

Question 3: Add the following attributes to the Student class with appropriate Setter and Getter functions:
‘studentNumber’, ‘majorName’, ‘universityName’

Answer: To add the ‘studentNumber’, ‘majorName’, and ‘universityName’ attributes to the ‘Student’ class with appropriate setter and getter methods, you can modify the class as follows:

```
public class Student extends Human {
    private String fullName;
    private String studentNumber;
    private String majorName;
    private String universityName;

    public Student(String fullName) {
        this.fullName = fullName;
    }
```

```
public String getStudentNumber() {
   return studentNumber;
}

public void setStudentNumber(String studentNumber) {
   this.studentNumber = studentNumber;
}

public String getMajorName() {
   return majorName;
}

public void setMajorName(String majorName) {
   this.majorName = majorName;
}

public String getUniversityName() {
   return universityName;
}

public void setUniversityName(String universityName) {
   this.universityName = universityName;
}
```

```
    public void sayMyName() {
        System.out.println("My name is " + fullName);
    }
}
```

With these changes, you can now set and get the ‘studentNumber’, ‘majorName’, and ‘universityName’ attributes of a ‘Student’ object using the appropriate setter and getter methods.

Question 4: Add the following attributes to the Professor class with appropriate Setter and Getter functions:
professorFaculty, numberOfCourse, professorSpecialty

Answer:To add the ‘professorFaculty’, ‘numberOfCourse’, and ‘professorSpecialty’ attributes to the ‘Professor’ class with appropriate setter and getter methods, you can modify the class as follows:

```
public class Professor extends Human {
    private String fullName;
    private String professorFaculty;
    private int numberOfCourse;
    private String professorSpecialty;

    public Professor(String fullName) {
        this.fullName = fullName;
    }

    public String getProfessorFaculty() {
        return professorFaculty;
```

```
}

public void setProfessorFaculty(String professorFaculty) {
   this.professorFaculty = professorFaculty;
}

public int getNumberOfCourse() {
   return numberOfCourse;
}

public void setNumberOfCourse(int numberOfCourse) {
   this.numberOfCourse = numberOfCourse;
}

public String getProfessorSpecialty() {
   return professorSpecialty;
}

public void setProfessorSpecialty(String professorSpecialty) {
   this.professorSpecialty = professorSpecialty;
}

public void sayMyName() {
   System.out.println("My name is " + fullName);
```

```
    }
}
```

With these changes, you can now set and get the ‘professorFaculty’, ‘numberOfCourse’, and ‘professorSpecialty’ attributes of a ‘Professor’ object using the appropriate setter and getter methods.

Question 5: Change the Professor class so that when the sayMyName() method is called on it, the fullName and professorFaculty attributes are printed in the output.

Answer:To modify the ‘sayMyName()’ method in the ‘Professor’ class so that it prints both the ‘fullName’ and ‘professorFaculty’ attributes, you can update the method as follows:

```
public class Professor extends Human {
    private String fullName;
    private String professorFaculty;
    private int numberOfCourse;
    private String professorSpecialty;

    public Professor(String fullName) {
        this.fullName = fullName;
    }

    public String getProfessorFaculty() {
        return professorFaculty;
    }
```

```
    public void setProfessorFaculty(String professorFaculty) {
        this.professorFaculty = professorFaculty;
    }

    public int getNumberOfCourse() {
        return numberOfCourse;
    }

    public void setNumberOfCourse(int numberOfCourse) {
        this.numberOfCourse = numberOfCourse;
    }

    public String getProfessorSpecialty() {
        return professorSpecialty;
    }

    public void setProfessorSpecialty(String professorSpecialty) {
        this.professorSpecialty = professorSpecialty;
    }

    public void sayMyName() {
        System.out.println("My name is " + fullName + " and I am from " +
professorFaculty);
    }
```

```
}
```

Now, when you call the ‘sayMyName()’ method on a ‘Professor’ object, it will print both the professor's full name and faculty.

Question 6: After completing the Student and Professor classes, create an instance of each and use the keyword instanceof to check whether these two instances are really an instance of the Human class or not.

Answer:To create instances of the ‘Student’ and ‘Professor’ classes and check if they are instances of the ‘Human’ class, you can use the instanceof keyword as follows:

```
public class Main {
    public static void main(String[] args) {
        Student student = new Student("John Doe");
        Professor professor = new Professor("Jane Smith");

        System.out.println("Is student an instance of Human? " + (student instanceof Human));
        System.out.println("Is professor an instance of Human? " + (professor instanceof Human));
    }
}
```

When you run this code, it will create instances of ‘Student’ and ‘Professor’ and then check if they are instances of the ‘Human’ class. Since both ‘Student’ and ‘Professor’ classes inherit from the ‘Human’ class, the output will be true for both instances.

Question 7: Write the facing code:
Human human = new Student();
What is the output of human.sayMyName()?

Answer:In Java, when you write `Human human = new Student();`, you are creating a reference of type `Human` pointing to an object of type `Student`. This is possible because `Student` is a subclass of `Human`, so it can be treated as a `Human` object.However, the method `sayMyName()` is not a method of the `Human` class, it's a method of the `Student` class. Since the reference `human` is of type `Human`, which does not have a `sayMyName()` method, you would get a compilation error if you try to call `human.sayMyName()` directly.

To call the `sayMyName()` method, you would need to cast `human` to a `Student` object:

Human human = new Student();

((Student) human).sayMyName();

This would output the student's name.

Question8: Now write this code:
Human human = new Professor();
What is the output of human.sayMyName()?

Answer: Similarly to the previous scenario, when you write `Human human = new Professor();`, you are creating a reference of type `Human` pointing to an object of type `Professor`, which is possible because `Professor` is a subclass of `Human`.However, just like before, the `sayMyName()` method is not a method of the `Human` class, it's a method of the `Professor` class. Therefore, you would need to cast `human` to a `Professor` object to call the `sayMyName()` method:

Human human = new Professor();

((Professor) human).sayMyName()

This would output the professor's name.

Question 9: What can be concluded from these two examples?

Answer: These two examples demonstrate the concept of polymorphism in object-oriented programming. Polymorphism allows objects of different classes to be treated as objects of a common superclass. In both examples, an object of a subclass (`Student` or `Professor`) is assigned to a reference variable of the superclass (`Human`). This is possible because the subclass objects are instances of the superclass due to inheritance.However, when it comes to method invocation, the actual method that gets called is determined by the type of the object, not the type of the reference variable. This is why, in both cases, you need to cast the reference variable to the subclass type in order to call the `sayMyName()` method specific to that subclass.

In summary, these examples illustrate how polymorphism allows for flexibility and code reusability in object-oriented programming by enabling different subclasses to be treated as instances of their common superclass.