1. We would like to design a class `Square` that inherits from `Rectangle`. A `Square` has the constraint that the four sides are of the same length. Consider the `Rectangle` class below:

```
1  /*
2      Public Class must be in its own file
3      with the filename the same as the class name
4  */
5  public class Rectangle {
6    private double width;
7    private double height;
8
9    public Rectangle(double width, double height) {
10     this.width = width;
11     this.height = height;
12   }
13
14   @Override
15   public String toString() {
16     return "Height: " + this.height + " Width: " + this.width;
17   }
18 }
```

   (a) How should `Square` be implemented to obtain the following output from JShell?

```
1  jshell> new Square(5);
2  $.. ==> Height: 5.0 Width: 5.0
```

   Suggested Guide:

```
1  public class Square extends Rectangle {
2    public Square(double length) {
3      super(length, length);
4    }
5  }
```

   The `toString` method is inherited from `Rectangle`.

   (b) Now implement two separate methods to set the width and height of the `Rectangle` class as follows:

```
1  public void setHeight(double height) {
2    this.height = height;
3  }
4  public void setWidth(double width) {
5    this.width = width;
6  }
```

   What undesirable design issues would this present?

Suggested Guide:

`Square` inherits the `setHeight` and `setWidth` methods from `Rectangle`. As a consequence, a square can be changed into a rectangle (*i.e.*, *the length of the sides no longer equal*).

```
jshell> Square s = new Square(5.0);
s ==> Height: 5.0 Width: 5.0
jshell> s.setHeight(4);
jshell> s;
s ==> Height: 4.0 Width: 5.0
```

```
class Vector2D {
    private double[] coord2D;
    // code omitted
}
```

(c) Now implement two overriding methods in the `Square` class as follows:

```
public void setHeight(double height) {
  super.setHeight(height);
  super.setWidth(height);
}
public void setWidth(double width) {
  super.setHeight(width);
  super.setWidth(width);
}
```

Do you think that it is now sensible to have `Square` inherit from `Rectangle`? Or should it be the other way around? Or maybe they should not inherit from each other?

Suggested Guide:

Based on the substitutability principle, if `Square` inherits from `Rectangle`, then anywhere we expect a `Rectangle`, we can always substitute it with a `Square`.

Consider the following example:

```
jshell> Rectangle[] rects = {new Rectangle(3.0, 5.0),
   ...>                      new Square(5.0)};
rects ==> Rectangle[2] { Height: 5.0 Width: 3.0, Height: 5.0
    Width: 5.0 }
jshell> rects[0].setHeight(4.0);
jshell> rects[0].setWidth(8.0);
jshell> rects[0];
$.. ==> Height: 4.0 Width: 8.0
jshell> rects[1].setHeight(4.0);
jshell> rects[1].setWidth(8.0);
jshell> rects[1];
$.. ==> Height: 8.0 Width: 8.0
```

Notice that setting `rects[1]` (*of type* `Rectangle`) to a height of `4.0` and a width of `8.0` does not produce the desired rectangle.

2. Given the following interfaces.

```
1  public interface Shape {
2    public double getArea();
3  }
4
5  public interface Printable {
6    public void print();
7  }
```

(a) Suppose the class `Circle` implements both interfaces above. Given the following program fragment,

```
1  Circle c = new Circle(new Point(0, 0), 10);
2  Shape s = c;
3  Printable p = c;
```

Are the following statements allowed? Why do you think Java does not allow some of the following statements?

i. `s.print();`

ii. `p.print();`

iii. `s.getArea();`

iv. `p.getArea();`

> **Suggested Guide:**
> Only `s.getArea()` and `p.print()` are permissible. Suppose `Shape s` references an array of objects that implements the `Shape` interface, so each object is **guaranteed** to implement the `getArea` method.
> Other than that, each object may or may not implement other interfaces (*e.g.*, *may not implement* `Printable`), so `s.print()` may or may not be applicable.
> In addition, we say that for the above statement `Shape s = c`, variable `s` has a compile-time type of `Shape` but a run-time type of `Circle`.

(b) Someone proposed to re-implement `Shape` and `Printable` as abstract classes instead. Would this work?

> **Suggested Guide:**
> No, you cannot inherit from multiple parent classes.

(c) Can we define another interface `PrintableShape` as follows:

```
1  public interface PrintableShape extends Printable, Shape {
2  }
```

and let the class `Circle` implements `PrintableShape` instead?

Suggested Guide:

Yes, it is allowed. Interfaces can inherit from multiple parent interfaces.

3. Using examples of overriding methods, illustrate why a Java class cannot inherit from multiple parent classes, but can implement multiple interfaces.

Suggested Guide:

If classes `A` and `B` have the same method `f()` defined, and class `C` inherits from them, which of the two parent methods will be invoked in `new C().f()`? However, for the case of two interfaces `A` and `B`, if they both specify `f()` to be defined by a class `C` that implements them, then an overridden method in `C` would satisfy both contracts.

In the case of interfaces with `default` methods, try compiling the program fragment below:

```
1  interface A {
2    default void f() { }
3  }
4
5  interface B {
6    default void f() { }
7  }
8
9  class AB implements A, B { }
```

Does it compile? What if only one of the interface's `f()` has a default implementation? Does it compile now?

If there are compilation errors, what are the compilation errors? What if an overriding method `f()` is implemented in class `AB`?