1 Inheritence

Whenever speaking of any object, we can make things more general (generalization) or more specific (specialization). For example, we say that a square is a shape—a shape is a more general description of a square. Similarly, we can claim that this shape is a circle, which is a more specific description of a shape.

Such relationships are generally classified under the broad concept of 'inheritence', and derive their meaning from one class extending another. For our shape example, the class Square would extend the Shape class.

```
public class Shape {
    // some code.
}

public class Square extends Shape {
    // some code.
}
```

The reason you might want to do this is that the 'parent' Shape class may have some functionality (that's common to all shapes—functionality such as drawing it, or finding an area, etc.) that you don't want to repeat in your 'child' classes.

In Java, you can only extend one class (each class can have exactly one parent). By default, every object you create has a java.lang.Object as parent. This is recursive, meanin that in our Square example, the Square is a Shape, and a Shape is an Object, meaning that the Square is also an Object.

There is implicit casting when 'down casting', or generalizing. For example, you can say:

```
Shape s = new Square(...);
```

After all, a square is a shape, so this should work without any prolems. You cannot do the reverse easily, for example, 'up casting', or specializing, ie: the below requires you explicitly cast the Shape into a Square, ie:

```
Shape s = new Square(...);
Square a = (Square)s;
```

The reason for this is because a shape can be anything, not just a square (imagine the shape is really a circle, and you're trying to use it as a square).