# Lecture 30

Static methods
Static variables

*Section 14.6*
Static Variables

```
public void start() {
     Student s1, s2, s3;

     s1 = new Student("Ann");
     s2 = new Student("Bob");
     s3 = new Student("Charles");

     System.out.println(s1.toString());
     System.out.println(s2.toString());
     System.out.println(s3.toString());
}
```

output

```
Ann, id: 0
Bob, id: 0
Charles, id: 0
```

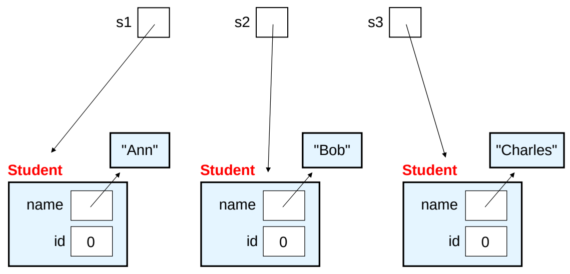
A Student object stores a name and an ID number.

How could the Student class be defined so that the output is as shown?

Solution

```
public class Student {

     private String name;
     private int id;

     public Student(String n) {
          name = n;
          id = 0;
     }

     public String toString() {
          return name + ", id: " + id;
     }
}
```

We would visualise the three Student objects as follows:



```
public void start() {
     Student s1, s2, s3;

     s1 = new Student("Ann");
     s2 = new Student("Bob");
     s3 = new Student("Charles");

     System.out.println(s1.toString());
     System.out.println(s2.toString());
     System.out.println(s3.toString());
}
```

output

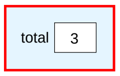
```
Ann, id: 1
Bob, id: 2
Charles, id: 3
```

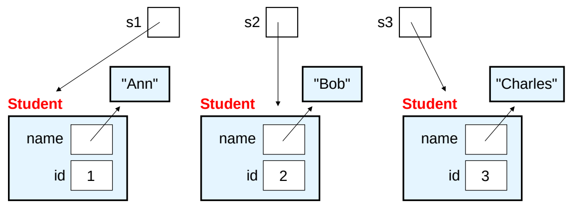
Now, define the Student class again.

This time, notice each student is assigned a unique ID number, starting from 1

## Having trouble?

- Each time the constructor method is called, we need to know how many *other* Student objects have already been created

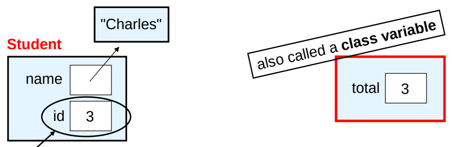




## static

- **static** is a modifier which can be applied to a variable or a method
- something that is **static** is associated with the class itself, and not with an instance of the class

## static variables



The ID number is associated with a specific *instance* or *object* of the Student class

The total is not associated with a specific Student object – it is information about the class in general

This is an ***instance variable*** of the Student class

This can be a ***static variable*** of the Student class

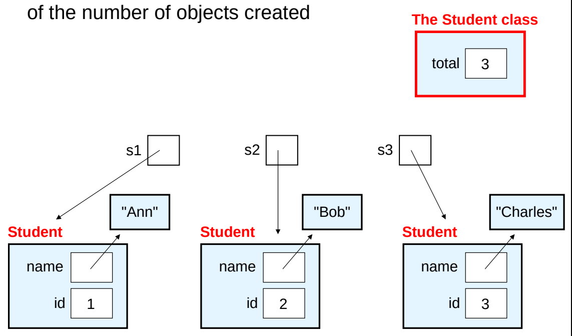
Solution

```
class Student {
      private String name;
      private int id;

      private static int total = 0;

      public Student(String n) {
            name = n;
            total++;
            id = total;
      }

      public String toString() {
            return name + ", id: " + id;
      }
}
```

This looks like an instance variable, but it is NOT. It is declared using the modifier static

The static variable is incremented each time an object is created

- There is one copy of the instance variables for every object created
- There is only ONE class variable, regardless of the number of objects created



## static methods

We are very familiar with ***instance methods***, and how to call them on objects:

We have also seen many examples of ***static methods***, or ***class methods***:

```
String s;
s = new String("abc");

int len = s.length();

System.out.println(len);
```

```
int max;

max = Math.max(10, 20);

System.out.println(max);
```

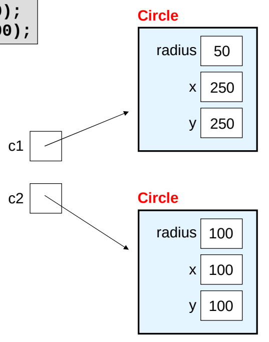
*Let's look at an example involving a Circle class...*

```
public class Circle {

      private int radius;
      private int x;
      private int y;

      public Circle(int r, int x, int y) {
            radius = r;
            this.x = x;
            this.y = y;
      }

      public void setSize(int r) {
            radius = r;
      }

      public String toString() {
            return new String("(" + x + "," + y + ")
                                          radius: " + radius);
      }
}
```

instance variables

instance methods

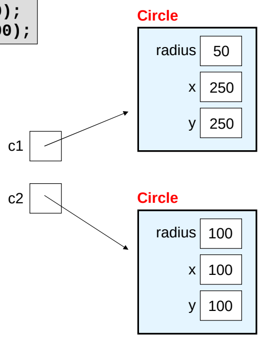
```
Circle c1, c2;

c1 = new Circle(50, 250, 250);
c2 = new Circle(100, 100, 100);
```



```
Circle c1, c2;

c1 = new Circle(50, 250, 250);
c2 = new Circle(100, 100, 100);
```

We have created two Circle objects. Each object is a specific instance of the Circle class

Each object has its own copy of the instance variables



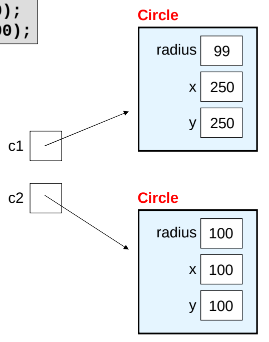
```
Circle c1, c2;

c1 = new Circle(50, 250, 250);
c2 = new Circle(100, 100, 100);
```

***Instance methods*** are called on a specific object. They use the instance variables of that specific object instance.

```
c1.setSize(99);
```

The Circle class

```
...
public void setSize(int r) {
        radius = r;
}
...
```



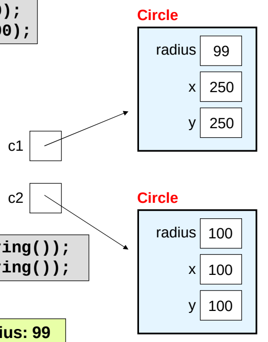
```
Circle c1, c2;

c1 = new Circle(50, 250, 250);
c2 = new Circle(100, 100, 100);
```

Although we call the *same* instance methods, we get *different* results because they are being called on different instances

```
System.out.println(c1.toString());
System.out.println(c2.toString());
```

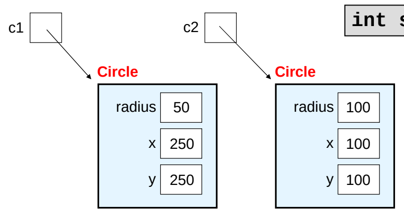
Output

```
(250,250) radius: 99
(100,100) radius: 100
```



## Which Circle is bigger?

- Let's say we have two Circle objects, and we would like to find out which one has the larger radius value.
- What method could we define in the Circle class to achieve this?



```
int size = ????
```

**we are going to define a method in the Circle class, how will we call it?**

```
public class Circle {

        private int radius;
        private int x;
        private int y;

        public Circle(int r, int x, int y) {
                radius = r;
                this.x = x;
                this.y = y;
        }

        public int biggerRadius(Circle other) {
                if (other.radius > radius) {
                        return other.radius;
                } else {
                        return radius;
                }
        }

        public void setSize(int r) {
                radius = r;
        }

        public String toString() {
                return new String("(" + x + "," + y + ")
                                                  radius: " + radius);
        }
}
```

**this is an instance method**

```
int size = c1.biggerRadius(c2);
```

```
public class Circle {

        private int radius;
        private int x;
        private int y;

        public Circle(int r, int x, int y) {
                radius = r;
                this.x = x;
                this.y = y;
        }

        public int biggerRadius(Circle a, Circle b) {
                if (a.radius > b.radius) {
                        return a.radius;
                } else {
                        return b.radius;
                }
        }

        public void setSize(int r) {
                radius = r;
        }

        public String toString() {
                return new String("(" + x + "," + y + ")
                                        radius: " + radius);
        }
}
```

here is another way we could write this instance method

but how would we *call* this method?

```
???? . biggerRadius(c1, c2);
```

```
public int biggerRadius(Circle a, Circle b) {
        if (a.radius > b.radius) {
                return a.radius;
        } else {
                return b.radius;
        }
}
```

this method does not rely on the state of any instance variables – all the information it needs is passed to it through the parameters

```
Circle c1, c2;

c1 = new Circle(50, 250, 250);
c2 = new Circle(100, 100, 100);
```

```
int size = c1.biggerRadius(c1, c2);
int size = c2.biggerRadius(c1, c2);
```

it is irrelevant which object the method is called on, because the method does not refer directly to any instance variables

```
Circle c1, c2;

c1 = new Circle(50, 250, 250);
c2 = new Circle(100, 100, 100);

int size = Circle.biggerRadius(c1, c2);
```

this method is associated with the Circle class, but not with a particular instance of the class

**we can use the name of the class to call the method as long as the method is declared with the modifier static**

```
public static int biggerRadius(Circle a, Circle b) {
        if (a.radius > b.radius) {
                return a.radius;
        } else {
                return b.radius;
        }
}
```

This is a static method, or class method:

```
public static int biggerRadius(Circle a, Circle b) {
        if (a.radius > b.radius) {
                return a.radius;
        } else {
                return b.radius;
        }
}
```

It is called using the name of the class:

```
Circle.biggerRadius( .... );
```

## static methods and variables

A static method ***cannot*** refer to any instance variables defined in the class

A static method ***can*** refer to static variables defined in the class

Let's return to the Student class example, and say we now want to print out the total number of Student objects that have been created:

```
Student s1, s2, s3;

s1 = new Student("Ann");
s2 = new Student("Bob");
s3 = new Student("Charles");

System.out.println(Student.totalStudents());
```

a static method has been defined in the Student class

```
class Student {
      private String name;
      private int id;

      private static int total = 0;

      public Student(String n) {
            name = n;
            total++;
            id = total;
      }

      public static int totalStudents() {
            return total;
      }

      public String toString() {
            return name + ", id: " + id;
      }
}
```

this static method can refer to the static variable total

```
class Student {
      private String name;
      private int id;

      private static int total = 0;

      public Student(String n) {
            name = n;
            total++;
            id = total;
      }

      public static int totalStudents() {
            return total + id;
      }

      public String toString() {
            return name + ", id: " + id;
      }
}
```

however this code would NOT compile, because a static method cannot refer to an instance variable

## Class constants

- Class constants are often useful
- A class constant uses the modifiers **static** and **final**
- It is OK to define a class constant **public**

```
public class Math {

     public static final double PI = 3.14159;
     ....
}
```

- We can refer to this in other classes as:

```
Math.PI
```

## Summary

Assume that class X contains the following method definitions:

```
public class X {
     ...
     public void yes() {...}
     public static void no() {...}
     ...
}
```

To call method yes():

```
X thing;
thing = new X();

thing.yes();
```

To call method no():

```
X.no();
```

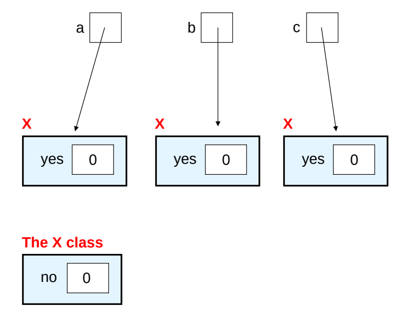
- static methods are also called *class methods*
- static methods are not allowed to refer to instance variables of the class in which they are defined (this should be obvious when you think about the fact that they are not called on any particular instance of the class), however they can refer to static variables
- static methods are mainly useful when you want to use methods without needing to create an object first (the Math class methods are good examples of this: eg. Math.max())

Assume that class X contains the following variable declarations:

```
public class X {
     private int yes;
     private static int no;
     ...
}
```

And assume the following objects are created:

```
X a, b, c;

a = new X();
b = new X();
c = new X();
```



- static variables are also called *class variables*
- there is only *one copy* of a class variable, regardless of how many objects are created
- an instance method in a class can refer to both the instance variables and the static variables defined in the class
- a static method in a class can only refer to static variables defined in the class
- static variables are commonly used to record how many instances of a class have been created