Classes comprise fields and methods

Fields:
- Things that describe the class or describe instances (i.e. objects)
- e.g. student number, first name, last name, gender, ...

Methods:
- constructors, getters, setters, other...
- e.g. getFirstName(), setFirstName(), equals()
A class has a name,

Fields
(called variables in Java code)

Methods
(A method may have local fields)
A quick look at two classes we have used: Math and Random

**Math** provides some useful utility methods. We use it without instantiating an object.

```java
double area = Math.PI * Math.pow(r, 2);
```

**Random** lets us use random sequences. To utilize this we must instantiate objects.

```java
Random die = new Random();
Int toss = die.next(6)+1;
```

<table>
<thead>
<tr>
<th>Math</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>+E</td>
<td>-seed</td>
</tr>
<tr>
<td>+PI</td>
<td>-multiplier</td>
</tr>
<tr>
<td>-Math()</td>
<td>+Random()</td>
</tr>
<tr>
<td>+abs(double a)</td>
<td>+Random(long seed)</td>
</tr>
<tr>
<td>+abs(float a)</td>
<td>+nextBoolean()</td>
</tr>
<tr>
<td>+ abs(int a)</td>
<td>+nextInt()</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>+ max(double a, double b)</td>
<td></td>
</tr>
<tr>
<td>+ max(int a, int b)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Math has two static fields

Math has a *private* constructor
You cannot instantiate a Math object

Math has many static methods

To use π you write
Math.PI

To use the static method `max` you write
Math.max(n1, n2)

*Here we specify the name of the class*
We must instantiate an object to get a random sequence

```java
Random gen = new Random();
```

Random has *some private* instance fields

- `seed`
- `multiplier = 0x5DEECE66DL`

### Constructors

- `Random()`
- `Random(long seed)`

### Instance methods

- `gen.nextBoolean()`
- `gen.nextInt()`

*gen* is an object ... an instance
Fields may be primitive variables
Or, they may be of some other type (e.g. String, ...)

May be public or private
  public – anyone can use it
  private – limited access
Methods

• Methods are either:
  – value-returning
    must have a return statement
    e.g. getters ⇐ naming convention is ...
  – void
    no return statement
    e.g. setters ⇐ naming convention is ...
• public vs private
  public - anyone can use it
  private - special cases
  Math constructor is private – you cannot instantiate a Math object ... try to do it
• All classes should have
  
  `equals(...)`

  `toString()`
equals Method

equals(...)

- Value-returning
- Returns a boolean
- Usually an equals method is designed for a class. Designer must determine the condition when two objects are considered equal.

- E.g. String class has an equals method

```java
string1.equals(string2)
“abc”.equals(“xyz”)  returns false
“abc”.equals(“abc”)  returns true
```
**toString Method**

**toString()**

- Value-returning
- Returns a string
- A method automatically called when an object is displayed
  
  E.g. `System.out.println(myObject);`
- The designer of a class determines what it returns
- E.g. `ArrayList` has a `toString()` method ... result is of the form:
  
  `[ object₁, object₂, ... objectₙ ]`
Consider a Word class

public class Word {
    String text;
    int frequency;

    public Word(String w){
        text = w;
        frequency = 0;
    }
}

Create Word in BlueJ and experiment.

Example:

public class ProcessWords {
    public static void main(String[] args) {
        // put your code here
        Word w = new Word("Java");
        System.out.println(w);
        System.out.println(w.text);
        System.out.println(w.frequency);
    }
}

What should be public?
What should be private?

If we have two words, how do we know if they are equal, or the same?

What happens if we try to print a word?
Consider the student class in the text ➔
Class Diagram for Student

<table>
<thead>
<tr>
<th>Name of class →</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>- id</td>
<td></td>
</tr>
<tr>
<td>- firstName</td>
<td></td>
</tr>
<tr>
<td>- lastName</td>
<td></td>
</tr>
<tr>
<td>- gender</td>
<td></td>
</tr>
<tr>
<td>- active</td>
<td></td>
</tr>
</tbody>
</table>

| Constructors → | +Student () |
|               | +Student (firstName, lastName, gender, active) |
|               | -nextId () |
|               | +getId () |
|               | +getLastId () |
|               | + getFirstName () |
|               | + getLastName () |
|               | + getGender () |
|               | + isActive () |
|               | + getMajor () |
|               | + setLastId (newLastId) |
|               | + setFirstName (newFirstName) |
|               | + setLastName (newLastName) |
|               | + setGender (newGender) |
|               | + setActive (newActive) |
|               | + setMajor (newMajor) |
|               | + toString () |
|               | + equals (s) |

- means there is no public access to the field or method

+ means there is public access to the method

show 1, 2, or 3 compartments/ info as needed
instance vs class
e.g. consider Student class
Which fields are class?
Which fields are instance?

```java
/**
 * A student.
 */
public class Student {
    // class fields
    private static int lastId;
    // instance fields
    private int id;
    private String firstName;
    private String lastName;
    private char gender;
    private boolean active;
    private Subject major;
    // first constructor, no arguments
    public Student()
    {
        id = nextId();
        // default values for a student:
        firstName = "unknown";
        lastName = "unknown";
        gender = '?';
        active = false;
    }
    // second constructor, four arguments
    public Student (String firstName, String
```
Java code for Student - fields

instance vs class

**Instance** ≡ **object**

**Static field** ≡ **class-level field**

There is only one `lastId` field. It is a class-level field that is shared by all Student instances.

There are `id`, `firstName`, `lastName`, `gender`, `active`, and `major` fields for each Student instance. So each student can have different values.
Java code for Student - fields

private vs public

private:
    only directly accessible from within the class/object, and from outside the class via getters/setters

public: accessible from anywhere

A design principle is to make fields private but give public access to the getters and setters
Java code for Student - constructors

```java
public Student()
{
    id = nextId();
    // default values for a student:
    firstName = "unknown";
    lastName = "unknown";
    gender = '?';
    active = false;
}

// second constructor, four arguments
public Student(String firstName, String lastName, char gender, boolean active){
    id = nextId();
    // when parameters and fields have the same
    // name they are distinguished this way:
    // a field name alone refers to the
    // a field name prefixed with "this."
    // refers to an object’s fields.
    this.firstName = firstName;
    this.lastName = lastName;
    this.gender = gender;
    this.active = active;
}
```

The no-arg constructor

Constructor with 4 parameters

Use as many constructors as your application requires. Constructors differ in the number and type of parameters.
Notice

Getters (also called accessors) for most private fields

```java
public String getFirstName()
{
    return firstName;
}

public String getLastName()
{
    return lastName;
}

public char getGender()
{
    return gender;
}

public boolean isActive()
{
    return active;
}
```

Naming convention:
Start with “get” followed by the field name but this starts with a capital letter

Naming convention for boolean:
Start with “is” followed by the field name but this starts with a capital letter
Notice

Setters (also called mutators) for most private fields

```java
public void setFirstName(String newFirstName) {
    firstName = newFirstName;
}
public void setLastName(String newLastName) {
    lastName = newLastName;
}
public void setGender(char newGender) {
    gender = newGender;
}
public void setActive(boolean newActive) {
    active = newActive;
}
```

Naming convention:
Start with “set” followed by the field name but this starts with a capital letter
private method nextId
Used to control the id assigned to a new student object

toString
Executes when a student is printed

equals
Tests two student objects to see if they are ‘equal’
Class is a template for objects

How are these shown in UML? (Figures 7.2, 7.4, 7.5)
Objects:

instantiated/created via `new` – lots of examples
also called an *instance* – so we can speak of instance fields/methods
How are these shown in UML? (Figures 7.3, 7.4)
Listing 8.1:

Creates two students
One using the no-arg constructors and setters
The other using a 4-arg constructor
/**
 * Create two student objects
 * using the two constructors
 */

public class UseConstructors {
{
   public static void main (String[] args){
      // first, with the no-arg constructor
      Student jill = new Student();
      // use setters to complete the student object
      jill.setFirstName("Jill");
      jill.setLastName("Lee");
      jill.setGender('F');
      jill.setActive(true);
      // now with the other constructor
      Student sam = new Student("Samantha", "Jones", 'F', true);
      // display the students
      System.out.println(jill);
      System.out.println(sam);
   }
}