Classes

Classes are the most fundamental concept in object-oriented programming — most object-oriented languages, including Java, could be called "**class-oriented**" instead.

A Class is like an object constructor, or a "blueprint" for creating objects. The code below declares a simple Java class named Vector:

```
class Vector {
    double x;
    double y;
}
```

A class is declared using the keyword class, and there must be braces around the class's declarations. This class declares two fields of type double, named x and y, meaning that every Vector object will have these two fields.

You should already know most of the basic ideas about classes, objects, fields, methods and constructors, but a summary is below:

- An **object** is an entity within a program, which has fields and methods.
- A **class** is a "blueprint" for a type of object, which declares what fields and methods those objects will have.
- A **field** is a variable which belongs to an object, and holds one item of data about that object. Fields are sometimes called "properties" or "attributes".
- A **method** is a way of interacting with an object; it's a function or procedure which belongs to an object, and does something when it's called.
- A **constructor** is like a method, but is only called when a new object is created. Normally, a constructor assigns some initial values to the object's fields.

If you're unfamiliar or unconfident with any of these concepts, now is an excellent time to review them from Computer Programming last year, when you studied them in Python.

Objects

To create a Vector object use the java `new Keyword` write `new Vector()`.

However, this is useless if you don't also declare a variable to hold a reference to the new object — you can't access an object without a reference to it. So, it's common to write something like this:

```
Vector vector = new Vector();
```

This statement:

1. Declares a variable named vector of type Vector,
2. Creates a new Vector object,
3. Assigns a reference to the new object to the variable vector.

Don't confuse the variable for the object: `vector` is not the name of the object, it's the name of a **variable** which holds a **reference** to the object. This matters because two variables can reference the same object.

Notice how Java's naming conventions help to understand this code — Vector is a class and vector is a variable.

Constructors

You may have noticed that the Vector class declared above has no constructor — or at least, there is no constructor declared. In fact, the class has a "**default constructor**" which assigns default values to x and y (that's 0.0 because they're of type double).

It is more convenient to have a constructor which assigns initial values the programmer chooses:

```
class Vector {
    double x;
    double y;

    Vector(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

Now the class has the constructor we declared, instead of the default constructor. When you create a new Vector object, you must provide values for x and y, and the fields will be set by the constructor. The constructor has the same name as the class, and the keyword this refers to the current object (i.e. the one being constructed).

Remember to recompile the class when you modify the code.

You can no longer write new Vector() to create an object with default values for x and y — now you must write something like this:

```
Vector vector = new Vector(2, 3);
```

This statement:

1. Declares a variable named vector of type Vector,
2. Creates a new Vector object,
3. Calls the constructor, which assigns 2 to the x field and 3 to the y field,
4. Assigns a reference to the new object to the variable vector.

Java vs. Python

We could write the Vector class in Python:

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

The difference is more than just syntax. In Java, we need to declare the fields x and y in the class before assigning to them in the constructor, but in Python we don't need to. Objects in Java only have fields if they're declared, but in Python you can add or delete fields whenever you want. For example, in Python we can give a Vector object a new field named z, and delete the field x:

```
>>> vector = Vector(2, 3)
>>> vector.z = 4
>>> vector.z
4
>>> del vector.x
>>> vector.x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Vector' object has no attribute 'x'
```

But in Java, the Vector class does not declare a field named z:

```
Vector vector = new Vector(2, 3);
vector.z = 4;
            Error: cannot find symbol - variable z
```

There is also no way to delete a field from an object in Java; you **always** know what fields and methods an object has, just by knowing the object's class. Because Java is statically-typed, this means the compiler can detect typos and errors (e.g. a misspelled field name) before the program is run.