Classes and Constructors
A constructor for a class has an access attribute (such as `public`) but no return type, not even `void`. As in Python and any other object-oriented language, the job of a constructor is to initialize the instance variables of the class.

In Java the name of a constructor is the same as the name of its class – class `Student` will have a constructor `Student( )`. 
You can have multiple constructors for a class as long as they have different arguments. For example, here is a class with three constructors:

```java
public class Person {
    String name;
    int age;

    public Person(String who, int a) {
        name = who;
        age = a;
    }

    public Person(String who) {
        name = who;
        age = 0;
    }

    public Person() {
        name = "bob";
        age = 69;
    }
}
```
We can simplify the use of multiple constructors and the names of constructor arguments with the keyword \textit{this}, which always refers to the current class. When used as a method, \textit{this} takes the place of one of the constructors of a class. So the second and third constructors of our Person class could be written

```java
Person( String who ) {
    this(who, 0);
}
Person() {
    this("bob", 69);
}
```

If we wrote the last constructor as `Person("bob", 69)` that would be an error.
Here’s another use of the word *this*. Class `Person` has an attribute `name`; in the constructor we used *who* for the corresponding argument. We could have used *name* for the argument to the constructor. Then inside the constructor *name* refers to the argument and *this.name* refers to the class variable.
Putting all of this together, here is how I would write this class:

```java
public class Person {
    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public Person(String name) {
        this(name, 0);
    }

    public Person() {
        this("bob", 69);
    }
}
```
Here are some additional methods for class Person:

```java
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

public void birthday() {
    age += 1;
}
```
Here is a main method:

```java
public static void main(String[] args) {
    Person x = new Person("bob");
    x.setAge(69);
    x.birthday();
    System.out.println(x.getAge());
}
```
Here is a subclass of Person. Note that the subclass extends the parent class.

```java
public class Student extends Person {
    double gpa;

    public Student(String name) {
        super(name); // calls the Person constructor
        gpa = 4.0;
        setAge(18);
    }
}
```
Here are some more methods of class Student:

```java
public double getGPA() {
    return gpa;
}
```

```java
public void setGPA(double g) {
    gpa = g;
}
```

```java
public static void main(String[] args) {
    Student x = new Student("Hermione");
    x.setAge(20);
    System.out.println(x.getName());
}
```

These are methods of class Student but not class Person. On the other hand, *all* methods of class Person are also methods of class Student.