Java Classes - Using your classes

How the classes you write are being used

What’s the use of classes?

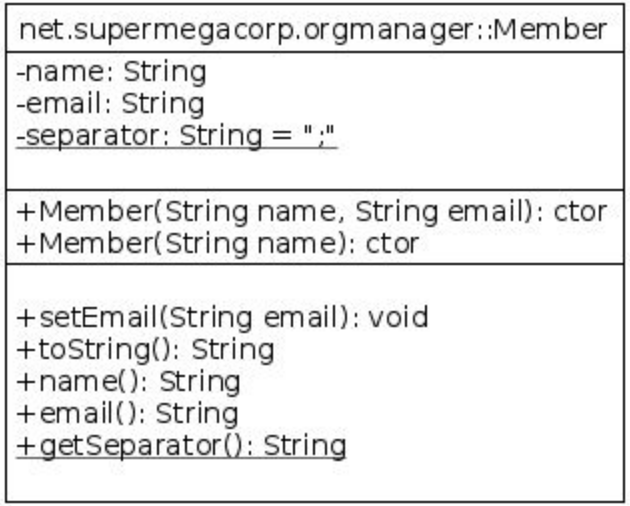
So, you have been writing a few classes by now… What for?

- The programs you will write will use objects to represent data and behavior
- The objects will be defined by various classes:
    - Classes from the Java standard API
    - Classes from other people
    - Classes you have written yourself
- You write classes when there is no predefined class available which suits your needs

Suppose you are writing a system for Members

You are writing a membership management system for a club.

You ended up designing a class representing a member in the registry:



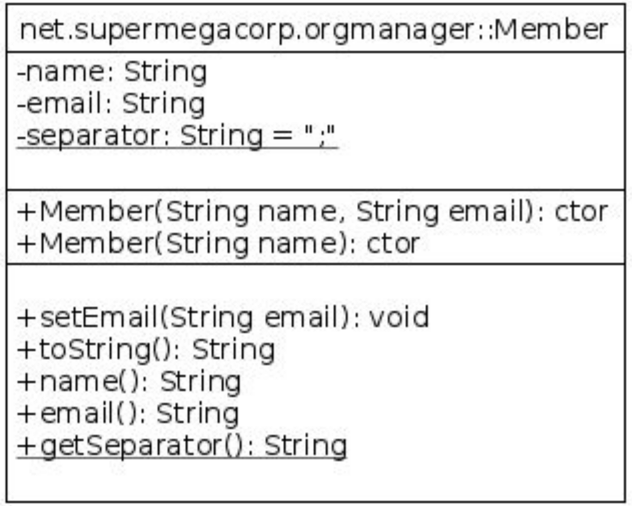
```
net.supermegacorp.orgmanager::Member
-name: String
-email: String
-separator: String = ";"

+Member(String name, String email): ctor
+Member(String name): ctor

+setEmail(String email): void
+toString(): String
+name(): String
+email(): String
+getSeparator(): String
```

The class Member

Two private instance variables name, email
a private static final String separator

Two constructors

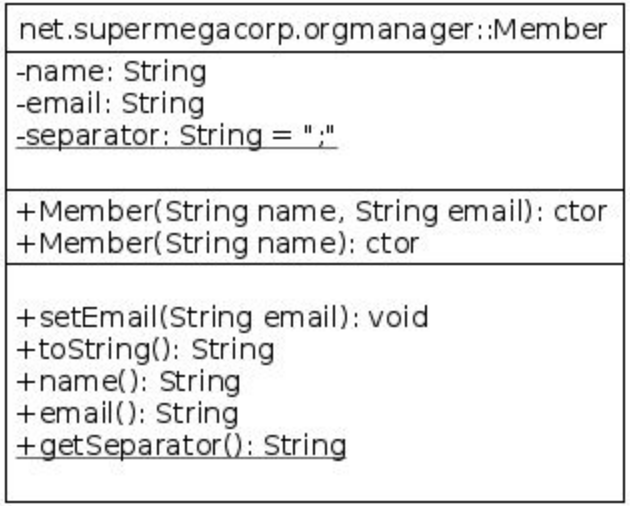
Some methods



```
net.supermegacorp.orgmanager::Member
-name: String
-email: String
-separator: String = ";"

+Member(String name, String email): ctor
+Member(String name): ctor

+setEmail(String email): void
+toString(): String
+name(): String
+email(): String
+getSeparator(): String
```

The class Member

```
package net.supermegacorp.orgmanager;

import java.io.Serializable;

public class Member implements Serializable {

  private String name;
  private String email;
  private static final String separator = ";";

  public Member(String name, String email) {
    this(name);
    setEmail(email);
  }

  public Member(String name) {
    this.name = name;
  }
// ... etc
```


net.supermegacorp.orgmanager::Member
-name: String
-email: String
-separator: String = ";"
+Member(String name, String email): ctor
+Member(String name): ctor
+setEmail(String email): void
+toString(): String
+name(): String
+email(): String
+getSeparator(): String

The use of the Member class

The Member class was written (by e.g. you) in order to have a class for objects in your system which represents members of the club.

In some module you might, for instance, have a method which adds a new Member to the club’s registry:

```
public void addMember(Member newMember) {
  memberList.add(newMember);
  save();
}
```

That’s one use of the Member class. You wrote it so that you could represent a member and pass it around in your system.

Using your own classes

Typically this is the steps:

- You analyse the problem (requirements etc)
  - You see that Member is a class which is used a lot in the Membership system
- You design a solution (design the classes etc needed to write the program)
  - You design the Member class so that it can fulfill its purpose in the system
- You write the code for the Member class
- You write the code for the classes and methods which need to use Member objects

Possible confusion for students new to Java

In Java, everything seems to be written inside a class. Sometimes the classes seem to simply represent some entity in the real world (e.g. a Member).

Other times classes are the units which starts the program (e.g. the class which contains the main() method)

Other times still, classes seem to represent stuff needed for the functionality of the system (e.g. some storage class, some GUI class etc)

So far in the book/course, we've mostly seen the first two types of classes:

- Classes that represents objects we use in the program to be
- Classes with the main() method

Possible confusion for students new to Java

One thing that might be confusing to you, is that when you write a simple class like the Member class, it contains a lot of code (constructors and methods) but you can’t really see the code which triggers those chunks of code!

Classes are in this sense “passive” descriptors. The contain the code for possible future objects, but those objects are created in some other class!

And the methods aren’t executed until someone has a reference to e.g. a Member object, and invokes a method via the reference. That too, is done from a different class.

Simple example showing client code and a class

```
// TestMember.java - client code using the Member class
// to create a Member object, save a reference to it
// and invoke a method via the reference
public static void main(String[] args) {
  Member testMember = new Member("Bengt", " bengan@klubben.se "); // hard coded values for testing
  System.out.println("The member email address: " + testMember.email() );
}

----------------------------------In a totally different file-----------------------------------
// Member.java - Model class - “passive” until some client code creates an object from it
public class Member {
  private String name;
  private String email;

  public Member(String name, String email) {
    this.name = name;
    this.email = email; // better yet, check that it seems like a real email first
  }

// ... etc etc
```

Summary

- The simple classes you write during this book/course typically represent objects needed in some application (a Member, a Car, a Contact, a Product...)
- The code which actually does something, starts in the main() method
    - The main method creates a few objects and calls methods on those objects via references
    - Often, an object’s method calls other methods (methods in the same object or methods of other objects (which might have been passed to the method as arguments)
- A class modelling a real-world thing is more of a blueprint for future objects which may be created by some code at some point
- Think of the class java.lang.String - it just sits there and does nothing until you or someone writes a program creating and using String objects!