Java
Object-oriented programming

What is OOP?

• Idea: User-defined types to complement primitive types like int, float...
• Definition of a new type is called a class. It contains:
  – Data
  – Methods: Code for performing operations on this data
• Example: the class String contains
  – Data: Sequence of characters
  – Operations: capitalize, substring, compare...
• Example: we could define a class Matrix with
  – Data: an m x n array of numbers
  – Operations: multiply, invert, determinant, etc.

Why OOP?

• Think of a set of classes as a toolbox:
  – You know what each tool does
  – You don't care how it does it
• OOP allows to think more abstractly:
  – Each class has a well defined interface
  – We can think in terms of functionality rather than in terms of implementation
• The creator of a class can implement it however he/she wants, as long the class fulfills the specification of the interface

A first example

```java
// The new type created is called sportTeam
class sportTeam {
  // The class a four members
  String homeTown;
  int victories, losses, points;
  public static void main(String[] args) {
    // we can declare variables of type sportTeam
    sportTeam Expos;
    // this create an object of type sportTeam and Expos now references it. */
    Expos = new sportTeam();
    Expos.victories = 62;
    Expos.homeTown = "Montreal";
    sportTeam Alouettes = new sportTeam();
    Alouettes.victories = 11;
  }
}

class sportTeam {
  String homeTown;
  int victories, losses, points;
  public sportTeam() {  /* see previous page */}
  public sportTeam(String town) {  /* see previous page */}
  public String toString() {
    return homeTown + ": " + victories + " victories, " + losses + " losses, for " + points + " points.";
  }
  public static void main(String[] args) {
    // now we can declare variables of type sportTeam
    sportTeam Expos, Alouettes;
    Expos = new sportTeam("Montreal");
    Expos.victories=62;
    Alouettes = new sportTeam("Montreal");
    Alouettes.victories = Expos.victories - 52;
    String report = Alouettes.toString();
    System.out.println(report);
  }
}
```

```java
// The new type created is called sportTeam
class sportTeam {
  // The class a four members
  String homeTown;
  int victories, losses, points;
  public static void main(String[] args) {
    // we can declare variables of type sportTeam
    sportTeam Expos, Alouettes;
    Expos = new sportTeam();
    Alouettes = new sportTeam("Montreal");
    Expos.victories=62;
    Alouettes.victories = Expos.victories - 52;
    String report = Alouettes.toString();
    System.out.println(report);
  }
}
```
Private vs public

- We don't want to let any part of a program access members of a class
  - It might disrupt the internal consistency of the object (e.g., one may increase the number of victories without increasing the number of points)
  - We want to hide as much as possible the inside of a class, to enforce abstraction.

Solution:
- Make these members private (they can only be used from inside the class)
- Allow access to these members only through predefined public methods

Assignments and equality testing

```
public static void main(String[] args) {
    sportTeam Expos, A louettes;
    sportTeam baseball, football;
    Expos = new sportTeam();
    A louettes = new sportTeam("Montreal");
    baseball = new sportTeam();
    A louettes.addWin();
    System.out.println("Alouettes.addWin()");
    System.out.println("Baseball = " + baseball.toString());
    football.addWin();
    System.out.println("Football = " + football.toString());
    System.out.println("Alouettes.addWin()");
    System.out.println("Football = " + football.toString());
    System.out.println("Baseball = " + baseball.toString());
}
```

Static members

- Normally, each object has its own copy of all the members of the class, but...
- Sometimes we want to have members that shared by all objects of a class
- The **static** qualifier in front of a member (or method) means that all objects of that class share the same member

This

- Sometimes, it can be useful for an object to refer to itself:
  - the **this** keyword refers to the current object
- We could rewrite the constructor as:

```
public class sportTeam {
    public String homeTown;
    // anybody can change the hometown
    private int victories, losses, points; // they can only be changed from within the class
    public sportTeam(String town) { /* see previous page */
    }
    public String toString() { /* see previous page */
        return "sportTeam: home = " + homeTown + ", sick = " + sickness + ", cure = " + cure;
    }
}
```

```java
public class sportTeam {
    public String homeTown;
    private int victories, losses, points;
    static public double exchangeRate; // all objects of type sportTeam share the same exchangeRate
    public sportTeam(String town) { /* see previous page */
    }
    public String toString() { /* see previous page */
        return "sportTeam: home = " + homeTown + ", sick = " + sickness + ", cure = " + cure;
    }
    public static void main(String[] args) {
        // now we can declare variables of type sportTeam
        sportTeam Expos, A louettes;
        sportTeam.exchangeRate = 1.57; // static members can be used without an actual object
        Expos = new sportTeam();
        A louettes = new sportTeam("Montreal");
        System.out.println("Rate from Expos: " + Expos.exchangeRate);
        System.out.println("Rate from A louettes: " + A louettes.exchangeRate);
    }
}
```