Classes in Java

- Classes
  - Fields (instance variables)
  - Constructors
  - Accessors & Mutators methods
  - toString() method
  - equals() method
  - Overload and Overwriting

Object-Oriented Programming

- O-O concentrates on objects (data)
- non O-O concentrates on procedures (methods)

- common terminology
  - abstraction
  - encapsulation
  - information protection (hiding)
  - inheritance
  - polymorphism

Data Encapsulation

- Why do we do this?
- think about the representation of dates
  - month (1-12)
  - days (1-31)
  - Year (positive)
- how do we encapsulate all its data?
  - Objects and classes

Classes

- Puts together the data and the operations of the object
- allows protections of data
  - Prevent accidents and errors
- allows inheritance
  - Code is reusable

Basic template of a Java Class

public Class Sample

{

Fields (class or instance variables)

Constructors

Methods

- accessors, mutators
- toString(), equals(), compareTo()
- other methods

}

Implementing a Class


```
class Date{
        private int day, month, year;

        Date();  //default constructor
        Date(int,int,int);

        public void setMonth(int);
        public int getMonth();
        public String toString();
}
```

Java Objects

- Data are called instance or class variables
  - Describe the state
  - Think on nouns
- Operations are called methods
  - Describe the behavior
  - Think of verbs

Building a Class

- Fields (instance variables) are private to prevent direct access and changes
- methods to be accessed from outside the class are public
- constructors are public
- methods to be accessed from within the class are private

Constructors

- initializes (instantiates) an object of a class
- it’s normal to have multiple constructors
- must be the same name as the name of the class
- they don’t return any values

- The “default” constructor has no parameters

Constructors

- Date(int d, int m, int y)

```
{
        day = d;
        month = m;
        year = y;
}
```

- Date()

```
{
        day = 1;
        month = 1;
        year = 2014;
}
```

Methods

- Some of them change the data
  - mutators or modification
  - usually of type void
  - set ...
- Some of them do not change the data
  - non-mutators or accessors
  - often they don’t have parameters
  - usually they return a value
  - get ...

Access Methods

- Any method that gives access to the private data members
  - getMonth()
  - getYear()
  - getDay()

- Accessors do not change the value of the private data, they are non-mutating functions.
- Accessors return the value of the private data members

Mutator Methods

- Any method that changes the private data members
  - setMonth()
  - setYear()
  - setDay()

- Modifier functions change the value of the private data, they are mutating functions.
- Most of the time they are of type void

toString() method

- Every Java class should have one
- Prints the relevant data of one object
- Uses a nice format
- It returns a String object

```
System.out.println(date1.toString());
System.out.println(date1);
```

equals() method

- It allows to compare two objects of the same class
  - It is a boolean method
- The implementation specifies the criteria that makes the two objects equal or not.
- It follows the same syntax as the equals() method of the String Class.

  object1.equals(object2)

equals() and toString() methods

- The signature of these methods is the same across all the classes in Java
- The signature consists of the method's name, and its parameters (number, type, order)

```
public boolean equals(ClassName object_name)
{ ….}

public String toSring()
{ ….}
```

Overriding methods in Java

- It means redefining a method in a superclass
- Must have same name, same type, and same formal parameter list (i.e. the same signature!)
  - equals()
  - toString()
  - compareTo()

Overloading methods in Java

- This is known as *polymorphism*
- Several methods, within the same class, with the same name and type, but different formal parameter list (i.e. different signatures!)
  - Constructors belong to this category

Static fields

- It is a field (data) that is shared by the entire class
- It is not a field of a particular object of the class

- In a class, RollDice, it makes sense to keep track of the number of rolls. But each object of the class (e.g. one roll) doesn't need this info.

- In a class, StudentRoster, it makes sense to keep track of the number of students. But each object of the class (e.g. one student) doesn't need this info.

Static methods

- Methods that belong to the class, not to a particular object of the class.

- Math Class is a typical example.

- They are not called by an object as in

  object.method → lastname.length()

but by the class itself, as in

  class.method → Math.sqrt(x)

Homework

- Homework #5 (interfaces & fractals) due tonight
- Quiz #5 (interfaces and abstract classes) tomorrow in recitation
- No homework due next week