Overloading

An ***overloaded*** name has more than one meaning.

Example: the **+** operator.

Example: println calls, with the method signature:

```
System.out.println(5)       println(int i)
System.out.println(true)    println(boolean b)
System.out.println(5.2)     println(double d)
System.out.println()        println()
System.out.println("hi")    println(String s)
```

A Student class

```
class Student {
  private String name;
  private String stunum;
  private int credits;
  private double GPA;

  // the usual kind of constructor
  public Student(String name, String stunum) {
    this.name = name;
    this.stunum = stunum;
    // What are the values of credits, GPA?
  }
}
```

Add another constructor

```
// constructor for transfer students
public Student(String name, String stunum,
    int credits, double GPA) {
  this.name = name;
  this.stunum = stunum;
  this.credits = credits;
  this.GPA = GPA;
}

// the "ordinary" constructor
public Student(String name, String stunum) {
  this(name, stunum, 0, 0.0);
}
```

Calling an overloaded method

```
// print "pass" for a mark
System.out.print("letter grade? (y/n) ");
if (in.readLine().equals("y")) {
  String mark = in.readLine();
  if (isPass(mark))
    System.out.println("pass");
}
else {
  int mark = Integer.parseInt
      (in.readLine());
  if (isPass(mark))
    System.out.println("pass");
}
```

Defining an overloaded method

```
private static boolean isPass (int mark) {
  return mark >= PASS_THRESHOLD; // 50?
}

private static boolean isPass (String mark) {
  return mark.compareTo(HIGHEST_FAIL_LETTER)
    < 0; // "E"?
}
```

This works nicely, but notice that we have to keep track of which marks are numerical grades and which are letter grades.

- What if you had a whole array of Vector of marks, some numerical and some letter?

A whole bunch of marks

```
Vector markList = new Vector();
while (true) {
  String mark = in.readLine();
  if (mark.equals("quit")) break;
  System.out.print("letter grade? (y/n) ");
  if (in.readLine().equals("y"))
    markList.addElement(mark);
  else
    markList.addElement(new Integer(mark));
}
```

But when we read it back, is it a String or an Integer?

- Let's write a Mark class that knows what it is.

A Mark class

```
class Mark {
  private boolean isLetterGrade;
  private String markString;
  private int markInt;

  public Mark(String mark, boolean isLetter) {
    isLetterGrade = isLetter;
    markString = mark;
    if (! isLetter)
      markInt = Integer.parseInt(mark);
  }
  public int intValue () {
    if (! isLetterGrade) return markInt;
    if (markString.equals("A")) return 88;
(etc.) ... }}
```

Using the Mark class

While filling the Vector of marks, we now do this:

```
  String mark = in.readLine();
  System.out.print("letter grade? (y/n) ");
  Mark m;
  if (in.readLine().equals("y"))
    m = new Mark(mark, true);
  else
    m = new Mark(mark, false);
  markList.addElement(m);
```

Using the Mark class again

Now let's average the marks stored in the Vector:

```
// Why don't we need a "count" variable?
int sum = 0;
for (int i = 0; i < markList.size(); i++) {
  Mark m = (Mark) markList.elementAt(i);
  sum += m.intValue();
}
if (markList.size() == 0)
  System.out.println("no good");
else
  System.out.println(((double)sum)/count);
```

Good, but messy in the long run

What we have works, but it requires the Mark class to be re-written every time we add a new kind of mark. (Pass-fail? Directly recorded grade point? A different scale for transfer credits?)

Instead, let's make a new kind of Mark for each kind of mark:

```
class LetterGrade extends Mark {
  private String mark;
  public LetterGrade (String mark) {
    this.mark = mark;
  }
  public int intValue () {
    if (mark.equals("A+")) return 95;
    if (mark.equals("A")) return 88;
    ...  } ... }
```

Whee!

```
class PercentGrade extends Mark {
  private int mark;

  public PercentGrade (String mark) {
    this.mark = Integer.parseInt(mark);
  }

  public int intValue () {
    return mark;
  }
}
```

What about Mark itself?

```
class Mark {
  public int intValue () {
    return -9999;  // We're hoping the student
                   // will complain if this
                   // mark is actually used.
  }
}
```

Mark is the ***parent class*** or ***superclass***.
PercentGrade and LetterGrade are ***child classes*** or ***subclasses***.

Using Mark again

```
String mark = in.readLine();
System.out.print("letter grade? (y/n) ");
Mark m;
if (in.readLine().equals("y"))

  m = new LetterGrade(mark);    // ?!

else

  m = new PercentGrade(mark);

markList.addElement(m);
```

Finding the average

```
int sum = 0;
for (int i = 0; i < markList.size(); i++) {
  Mark m = (Mark) markList.elementAt(i);

  sum += m.intValue(); // ?! again

}
if (markList.size() == 0)
  System.out.println("no good");
else
  System.out.println(((double)sum)/count);
```

Another inheritance example

Let's define a Student as a kind of Person. First, what's a Person?

```
class Person {
  private String name;
  private String birthdate;
  public Person (String name, String bdate) {
    this.name = name;
    this.birthdate = bdate;
  }
  public String getName () { return name; }
  public String toString() {
    String result = "Name: " + name;
    result += "\nBirthdate: " + birthdate;
    return result
  }
}
```

An array of Persons

using Person in main( ):

```
int size = 10;
Person[] list = new Person[size];
for (int i = 0; i < list.length; i++) {
  String name = in.readLine();
  String bdate = in.readLine();
  list[i] = new Person(name, bdate);
}

for (int i = 0; i < list.length; i++)
  System.out.println(list[i]);
```

A Student is a kind of Person

```
class Student extends Person {
  private String stunum;
  public Student (String name, String bdate,
      String stunum) {
    // How do we set name and birthdate?
    this.stunum = stunum;
  }
  public String getStudentNumber () {
    return stunum;
  }
  public String toString () {
    String result = ...; // same as in Person
    result += "\nStudent number: " + stunum;
    return result;
  }   }
```

How is Student related to Person?

This works:

```
  Student stu = new Student("Jim", "1900",
      "0101");
  System.out.println(stu.getName());
```

The getName( ) method is ***inherited*** from Person.

What if we want to add eye colour as an attribute of Persons?
We'll need to change toString( ):

```
  result += "\nEye colour: " + eyeColour;
```

– and the same in Student (and every other child class).
We can use the keyword ***super*** to refer to Person's methods:

```
 String result = super.toString();
```