CMSC 131

Object-Oriented Programming I

Classes Introduction II

Dept of Computer Science
University of Maryland College Park

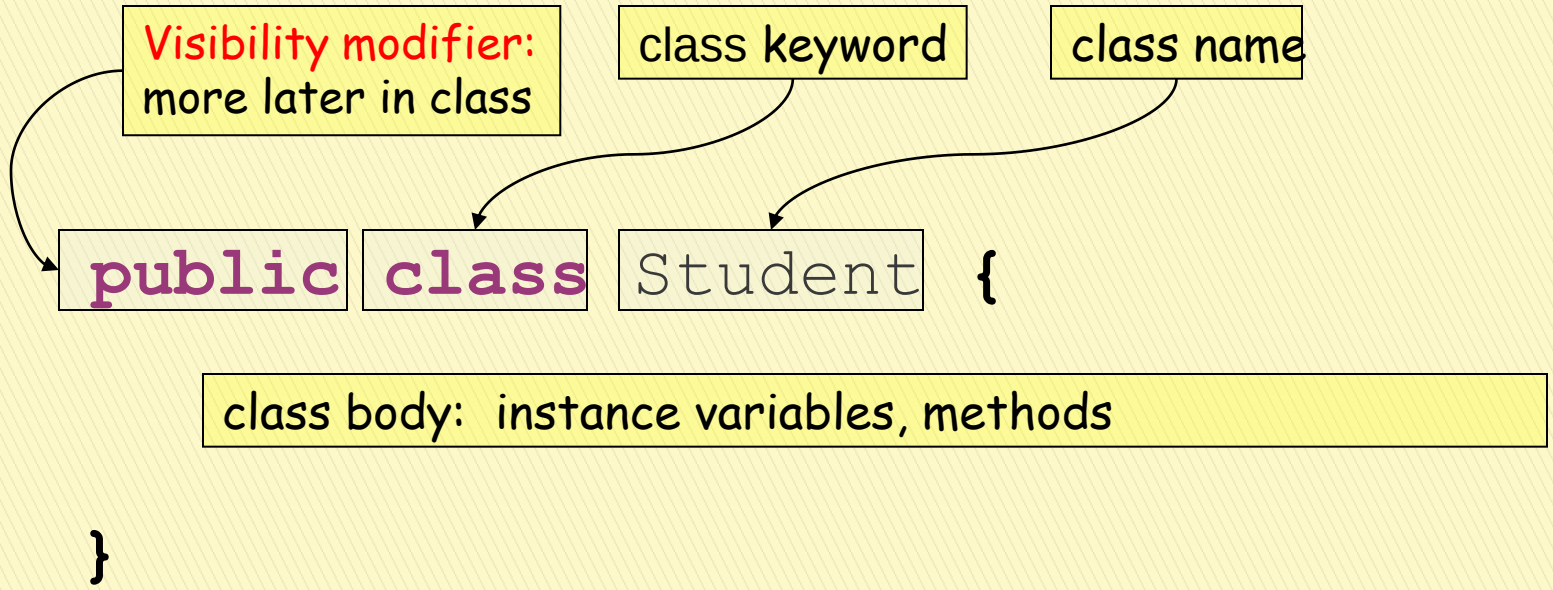
This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane

Overview

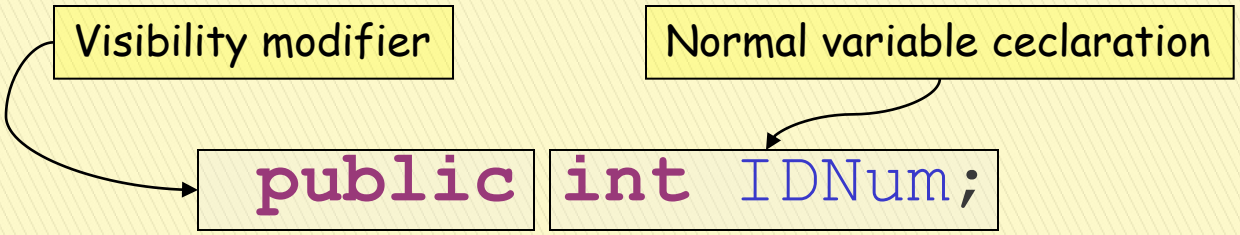
- Class Declaration
- Equals
- toString method

Classes in Java

- Class declarations have the following form in Java:



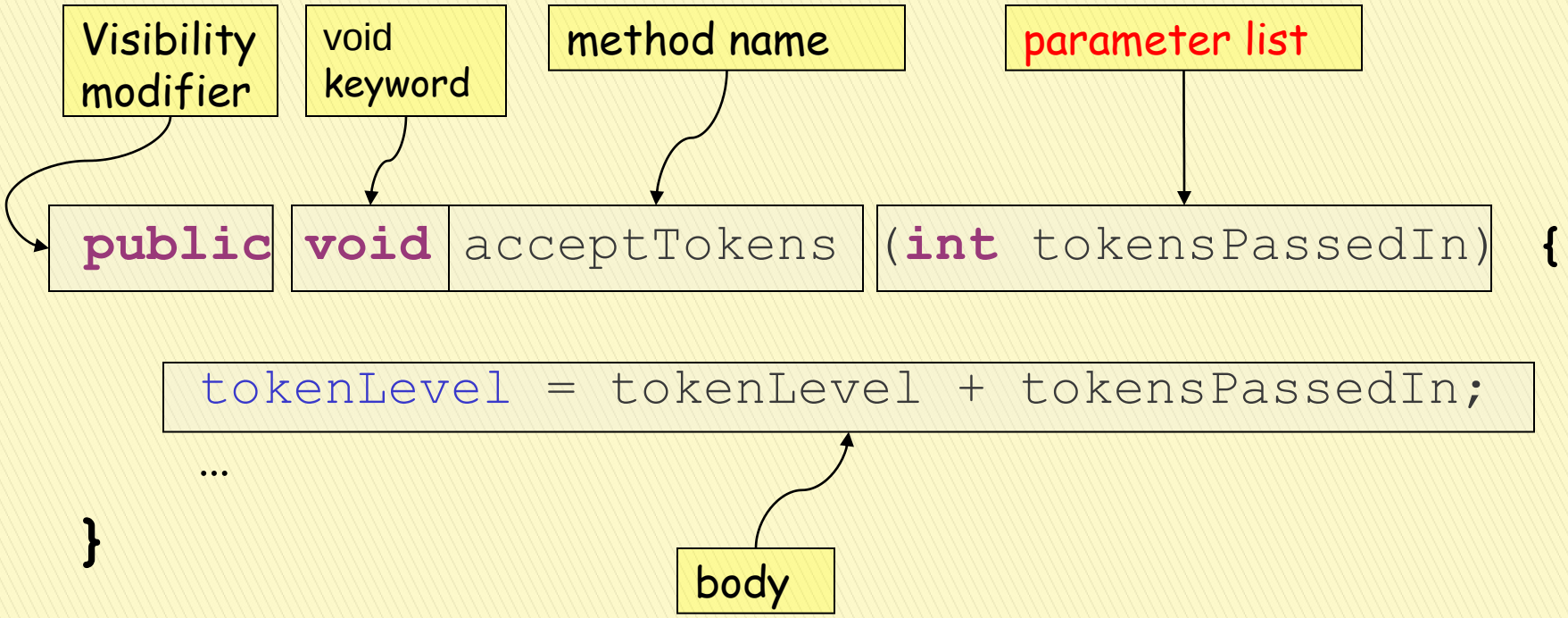
Anatomy of an Instance Variable Declaration



- We will see later that we can have private as visibility modifier
- What is the default value?
  - Boolean → false
  - Number → 0
  - Reference → null
- null → represents “no address”

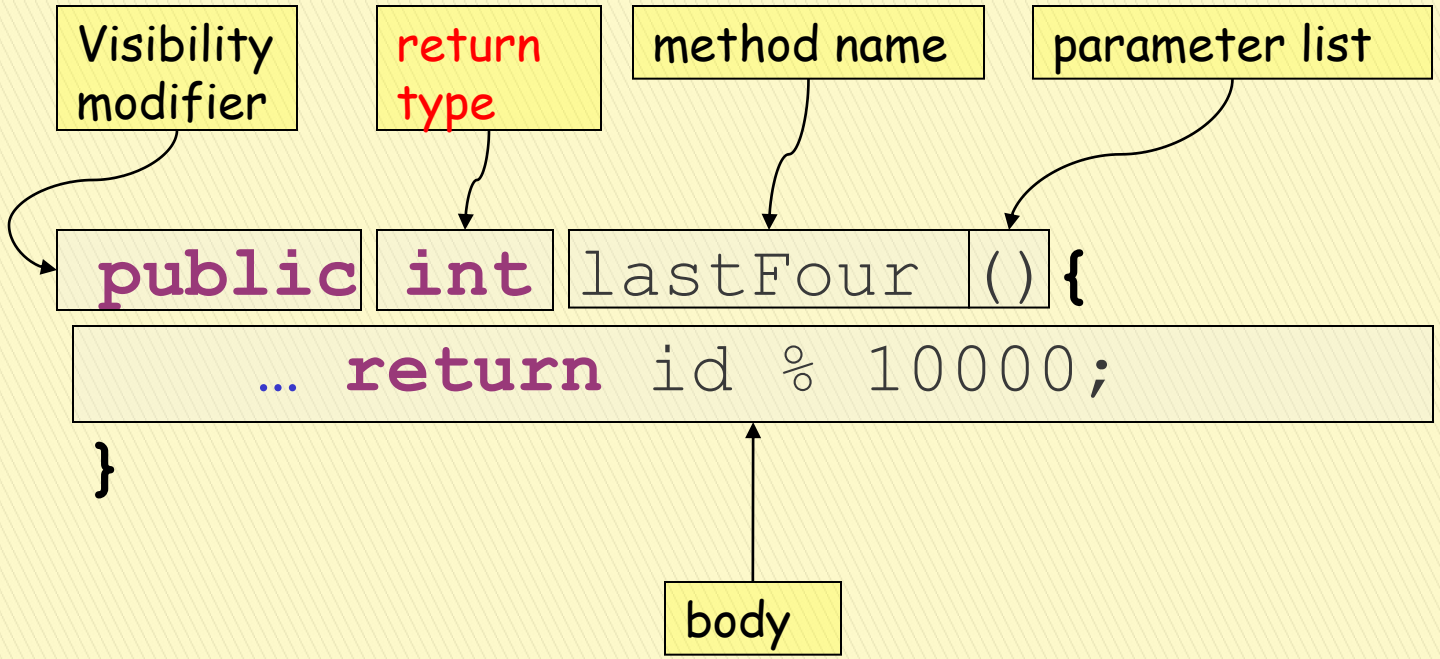
Anatomy of a Method Declaration for …

methods that do not return values



Anatomy of a Method Declaration for …

methods that return values



Return Type

- Methods that return values must specify the type of the value to be returned
- The bodies of these methods use `return` to indicate when a value is to be returned
- The value being returned must have the same type as the return type
- Notice that return can be used anywhere in the method
- For a method with no return type “return;” will end the method

Object Creation

- Once a class is defined, objects based on that class can be created using new:

  new Student()

- We are "creating an instance of class Student"
- To assign an object to a variable, the variable's type must be the class of the object

  Student s = new Student();
- Each object has its own copies of all the instance variables in the class
- Instance variables and methods in an object can be accessed using "." or using setter (mutator) methods

  s.IDNum = 123456789;

  s.setIDNum(123456789);

Constructors

- Special "methods" in class definitions to specify how objects are initialized
- Form of a constructor definition:

```
Student (String nameDesired, int IDDesired) {
    name = nameDesired;
    id = IDdesired;
}
```

- Can have more than one constructor, provided argument lists are different

```
Student (int IDDesired) {
    id = IDDesired;
}
```

- Java includes *default* constructor (no arguments), which you can redefine (overriding the default)

```
Student () {
    tokenLevel = 3;
}
```

Equality Testing

```
public boolean equals(Student otherStudent) {
    if (otherStudent == null) {
        return false;
    } else if (id == otherStudent.id) {
        return true;
    } else {
        return false;
    }
}
```

IMPORTANT: For now we will have a parameter different from Object but the correct approach to define equals is to use Object as a parameter

Objects to Strings

- What happens if we try to print a Student object?
  - Invoke println using a Student object as an argument?

    Student s1 = new Student ();

    System.out.println (s1);
- Something like this prints:

  Student@82ba41

Java Knows "How" To Print Any Object

- Why?
  - Every class has a default toString method
  - toString converts objects into strings
  - System.out.println calls this method to print an object
  - Default: object type and address

- toString can be overridden!

```
// The method for converting Students to strings

public String toString() {
    return (name + ": " + id);
}
```

Example

- Let's expand our SuperHero class
- Add two constructors
  - One with name and strength as parameters
  - One with just name
  - One with no name
- Let's add a set method
- Let's add a toString() method
- Let's add an equals method
- Let's add a method that takes an int and increases the strength
- Let's add a method that given a number of villains tells us whether the SuperHero can defeat them.
- Suggest your own … ☺