COMP 110-001
More About Classes

Yi Hong

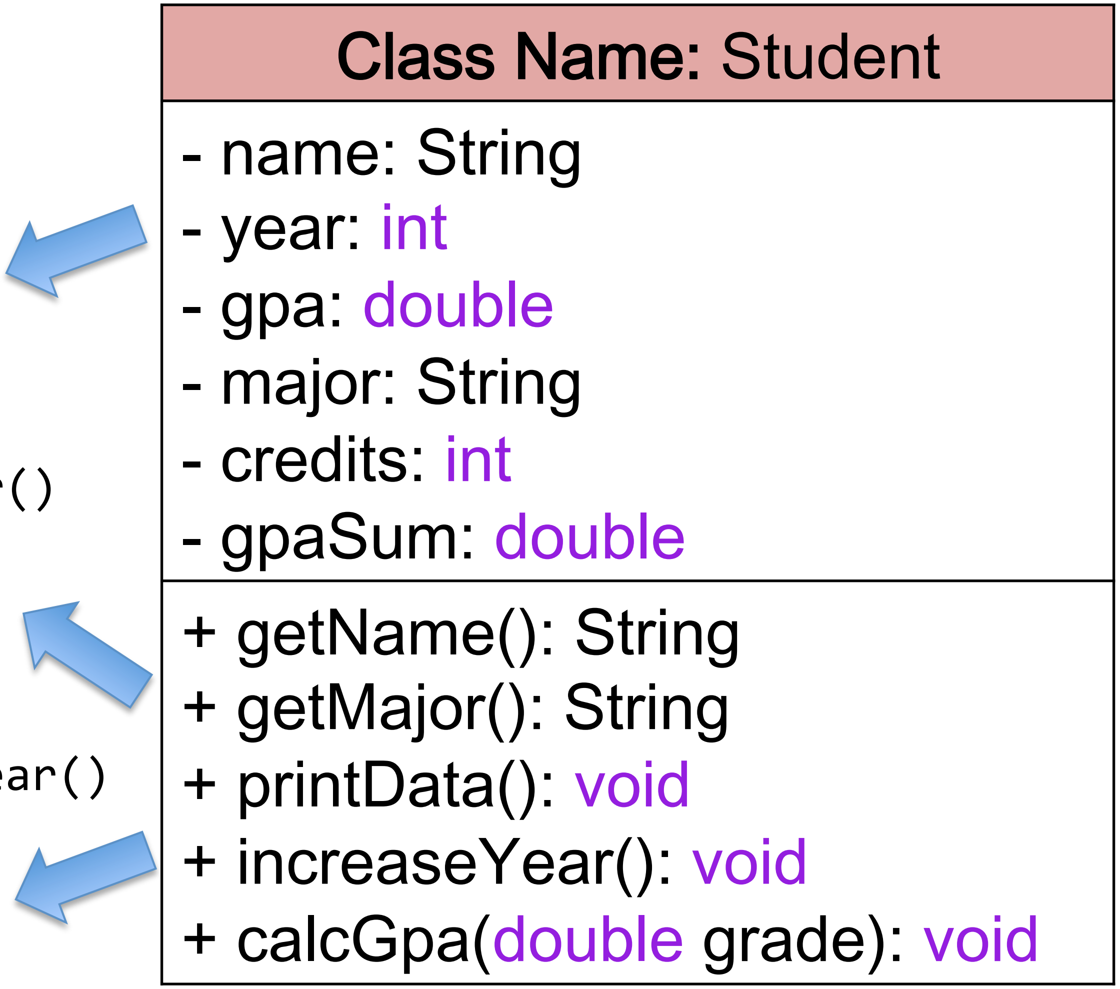
May 22, 2015

Review

```java
public class Student
{
    public String name;
    public int classYear;
    public double gpa;
    public String major;
    // ...

    public String getMajor()
    {
        return major;
    }

    public void increaseYear()
    {
        classYear++;
    }
    // ...
}
```



Today

- Methods
- Code block and variable scope

Classes, Objects, and Methods

- Class: a definition of a kind of object
- Object: an instance of a class
  - Contains instance variables (data) and methods
- Methods
  - Performs actions defined by a set of statements

Methods

- Two kinds of methods
  - Methods that return a value
    - Examples: String’s *substring()* method, String’s *indexOf()* method, etc.
  - Methods that return nothing
    - Perform some action other than returning an item
    - Example: System.out.println()

Methods

```
public String getMajor()
{
    return major;
}

public void increaseYear()
{
    classYear++;
}
```

returns a String

return type

returns nothing

Defining Methods That Return Nothing

```
public void increaseYear()
{
    classYear++;
}
```

- Method heading:
  - public: no restriction on how to use the method (more details later)
  - void: a void method that returns nothing
  - Method name
  - Method parameters (no parameters in this example)
- Method body: statements executed when the method is called (invoked)
  - **Must be inside a pair of braces {}**

Example: Method printData

- As usual, inside a block (defined by braces), you can have multiple statements

```
public void printData()
{
    System.out.println(“Name: ” + name);
    System.out.println(“Major: ” + major);
    System.out.println(“GPA: ” + gpa);
}
```

Example of Method with Parameters

```
public void increaseYear(int increment)
{
   classYear += increment;
}
```

Data type

Name of parameter

```
public void increaseYear(int increment, boolean check)
{
   if (check && classYear + increment <= MaxYear ) {
         classYear += increment;
   }
}
```

- Parameters are used to hold the values that you pass to the method
- Multiple parameters are separated by comma

Calling Methods That Return Nothing

- Object, followed by dot, then method name, then ()
  - Order, type, and number of arguments must match parameters specified in method heading
- Use them as Java statements

```
Student jack = new Student();
jack.classYear = 1;

jack.increaseYear();

System.out.println("Jack's class year is " + jack.classYear);
```

Defining Methods That Return a Value

```
public String getMajor()
{
    return major;
}
```

- Method heading:
  - public: no restriction on how to use the method (more details later)
  - Type: the data type of value that the method returns
  - Method name & parameters
- Method body: statements executed
  - **Must be inside a pair of braces {}**
  - **Must have a return statement**

return Statement

- A method that returns a value must have *at least one* return statement
- Terminates the method’s execution, and provides a value returned by the method. More statements follow the return statement will not be executed
- Syntax:
  - return Expression;
- Expression can be any expression that produces a value of the type specified in the heading of the method

Methods That Return a Value

- Example:

```
public String getClassYear()
{
    if (classYear == 1)
        return “Freshman”;
    else if (classYear == 2)
        return “Sophomore”;
    else if ...
}
```

- A better one:

```
public String getClassYear()
{
    String str = “”;
    if (classYear == 1)
        str = “Freshman”;
    else if (classYear == 2)
        str = “Sophomore”;
    else if ...
    return str;
}
```

Calling Methods That Return a Value

- Object, followed by dot, then method name, then () (the same as before)
- Use them as a *value* of the type specified by the method’s return type

```
Student jack = new Student();
jack.major = “Computer Science”;

String major = jack.getMajor();

System.out.println(“Jack’s full name is ” + jack.getName());
System.out.println(“Jack’s major is ” + major);
```

return Statement

- Can also be used in methods that return nothing
- Terminates the method
- Syntax:
  - Return;

```
public void increaseYear()
{
    if (classYear >= 4)
        return;
    classYear++;
}
```

Summary of Method Definitions

- Syntax

```
public Return_Type Method_Name(Parameters)
{
    Statements
}
```

  - Return_Type
    - void (don't need a return statement, but it can have one if you want to end the method invocation before the physical end of the code: return;)
    - a data type (Statements must contain at least one return statement of the form: return Expression;)

Calling Methods from Methods

- In a method's body, we can call another method
  - receiving_object.method();
- If calling a method in the same class, we do not need receiving_object:
  - method();
- Alternatively, use the this keyword
  - this.method();

this

- Within a class definition, this is a name for the receiving object
- The object is understood to be there, but its name usually is omitted
  - this.name
  - this.major
  - this.getMajor()
- See textbook p.282 for details

Code Block

- A section of code enclosed by { … }
- For grouping purpose

```
if ( x < 0 )
{
        isPositive = false;
        x = -x;
}

for(int i = 0; i<10; i++)
{
        System.out.println(“*”);
}
```

Code Block

- Code blocks can be nested

```
public class Hello
{
    public static void main(String arg[])
    {
        System.out.println("Hello.");          Outer block
    }
}
```

```
public class Hello{
    public static void main(String arg[])
    {
        System.out.println("Hello.");          Inner block
    }
}
```

Another Example of Code Block

```
for(int i = 0; i<100; i++)
{
    if ( i % 2 ==0 )

    {
        System.out.println(i + " is even");              Outer block
    }
}
```

```
for(int i = 0; i<100; i++)

{
    if ( i % 2 ==0 )

    {
        System.out.println(i + " is even");            Inner block
    }
}
```

Variable Scope

- The scope of a variable is the part of the program over which the variable name can be referenced
- Variables here include local / instance variables, and method parameters
- Two rules:
  - You cannot refer to a variable before its declaration
  - Variables defined in a block are only accessible within the block

Variable Scope

- Rule 1: (Use after definition)

```
s.nextInt();                                   ✗
.....
Scanner s = new Scanner(System.in);

i = 6;      ✗
......
int i;
```

Variable Scope

- Rule 1: (Use after definition)
  - Method parameters (ready for use in the method body)

```
public Color getColorFromString( String input ) {
    // input is available for using in the whole method body
    …
}
```

Variable Scope

- Rule 2: Variables defined in a block are only accessible within the block

```
int outer = 1;

{
  int inner = 2;
  System.out.println("inner = " + inner);
  System.out.println("outer = " + outer);
}

System.out.println("inner = " + inner);  ✗
System.out.println("outer = " + outer);
```

Cannot reference inner here

Variable Scope

- Rule 2: Variables defined in a block are only accessible from within the block

```
int outer = 1;

{
  int inner = 2;
  System.out.println("inner = " + inner);
  System.out.println("outer = " + outer);
}

int inner = 3;   // why I can define inner again?
System.out.println("inner = " + inner);
System.out.println("outer = " + outer);
```

```
inner = 2
outer = 1
inner = 3
outer = 1
```

Variable Scope

- What is the scope of instance variables?

Scope

```
public class Student
{
    public String name;
    public int classYear;
    public double GPA;
    public String major;
    // ...

    public String getMajor()
    {
        return major;
    }

    public void increaseYear()
    {
        classYear++;
    }
}
```

Revisit Local and Instance Variables

```
public class Student
{
    public String name;
    public int classYear;
    public String major;
    public void printInfo()
    {
        String info = name + ": " + major + ": " + classYear ;
        System.out.println(info);
    }

     public void increaseYear(int inc)
     {
         classYear += inc;
         String info = "classYear updated";
         System.out.println(info);
    }
}
```

Next Class

- No Class next Monday (Memorial Day)
- We will have Lab 4 & 5 next Tuesday
- Homework 2 due next Tuesday