

Lecture V:

Classes + Designing Complex Programs

Mark Huiskes

LIACS, Leiden University

Introduction to Programming, Media Technology MSc



Lecture Schedule

15 Sep	Lecture I	Variables, data types, operators + assignments
Today	Lecture II	Functions + mouse interaction
29 Sep		no class
6 Oct	Lecture III	Conditions + loops
13 Oct	Lecture IV	Arrays
Today	Lecture V	Classes + designing complex programs
27 Oct	Lecture VI	Recursion, data structures + sorting
3 Nov		no class
10 Nov	Lecture VII	Images and libraries
17 Nov	Lecture VIII	Max/MSP/Jitter (Edwin van der Heide)

Today

1. Tips + tricks
2. Summary sketch
3. Assignments last week
4. Classes + objects Basics
5. Background on object-oriented programming

Tips & Tricks

A few programming websites

News/popular links

- Hacker News: <http://news.ycombinator.com/>
- Proggit: <http://reddit.com/r/programming>
- Slashdot: <http://slashdot.org/>;
- Ars Technica: <http://arstechnica.com/>

Forums

- Processing.org Discourse
- Other languages:
 - <http://stackoverflow.com>
 - : <http://ask.slashdot.org/article.pl?sid=08/09/17/224230>
 - http://www.perlmonks.org/index.pl?node_id=511715
- IRC channels <http://searchirc.com>

Summary Sketch

Assignments last week

Classes + objects

```
Ball b1; // declare object
b1 = new Ball(20, 50, 255); // 'construct' the object
b1.draw() // 'call a method' of the ball
```

```
class Ball { // define class
    // properties
    int posX, posY, radius;

    // constructor
    Ball(int x, int y, int r) {
        posX = x;
        posY = y;
        radius = r;
    }

    // methods
    void draw() {
        ellipse(posX, posY, 2 * radius, 2 * radius);
    }
}
```

Classes and objects

daily life...

- Names for things can be understood at two levels:
 - as **abstract types** (e.g. the set of all chairs)
 - as **concrete objects**: (e.g. a particular chair)

object-oriented programming...

- **Classes** define data types, representing an **abstract type/category/concept**
- **Objects** are *instances of a class*, i.e. variables of the given type, representing the **concrete objects**
- Idea: model the problem you are solving with classes and objects.
- For that we describe the relevant **properties** and **functions/behavior** of the objects.

Properties

- Properties model object characteristics and object state
- In reality there tend to be many possible properties
- In our programs we only represent the ones that are useful for the problem...
- ...by putting property variables in the class

- Example: Employee class

Application 1: SimCompany game

properties: name, gender, height, hairColor, shoeStyle, location, speed

Application 2: Implementation database salary admin

properties: employeeNr, name, jobTitle, department, salary, holidaysLeft

Functions/Behavior

- Implemented through **methods**
- "method of doing the job": often "an object's way of providing a service"
- Again we only implement methods that are useful for the problem
- By putting functions in the class

- Example: Employee class

Application 1: SimCompany game

methods: `void appearBusyBehindDesk() {...}`
`void runAroundInTheBuilding(float speed) {...}`
`void joinMeeting(float gesturesPerMinute) {...};`

Application 2: Implementation database salary admin

methods: `void report(int nLastMonths);`
`void updateSalary(float newSalary);`
`void retire();`

```
Ball b1; // declare object
b1 = new Ball(20, 50, 255); // 'construct' the object
b1.draw() // 'call a method' of the ball
```

```
class Ball { // define class
    // properties
    int posX, posY, radius;

    // constructor
    Ball(int x, int y, int r) {
        posX = x;
        posY = y;
        radius = r;
    }

    // methods
    void draw() {
        ellipse(posX, posY, 2 * radius, 2 * radius);
    }
}
```

Class definition – general form

```
class Class-name {  
    property-declarations  
  
    // constructor(s)  
    Class-name ( arguments ) {  
        constructor-statements  
    }  
  
    method-definitions  
}
```

Defining a class – example

```
class Ball {  
  
    // properties  
    int posX, posY, radius;  
  
    // constructor  
    Ball(int x, int y, int r) {  
        posX = x;  
        posY = y;  
        radius = r;  
    }  
  
    // methods  
    void draw() {  
        ellipse(posX, posY, 2 * radius, 2 * radius);  
    }  
}
```

Constructor

- A block of code that gets activated at the creation of a new object
- Mainly to **initialize** (and possibly allocate) the **object properties**
- **No return type**, not even void
- Always has exactly the **same name as the class**
- There can be more than one constructor
 - always the same name, but with different argument types (overloading)
 - Just like for regular functions/methods: which constructor is used depends on the number of arguments (and arguments types) that the constructor is called with

Declaration and allocation of objects

- Declaration: like for built-in types

Class-name variable-name(s);

Examples: `Employee e1;`

`Employee e2, e3, randomGuy;`

- Allocation: reserving memory for the object

variable-name = new constructor-call;

Examples: `e1 = new Employee("John", 58);`

`e2 = new Employee();`

- Or both at once: `Employee e1 = new Employee("John", 58);`

Using Objects: dot notation

- Accessing properties
object-variable.property

Examples: `println(e1.name);`
 `float salaryMay = e1.salary + holidayMoney;`
 `e1.department = "R&D";`

- Calling methods
object-variable.method-name(arguments)

Examples: `e1.report(5);`
 `randomGuy.newSalary(100000);`
 `randomGuy.runAroundInBuilding(6);`

Example V.1

Accessing properties and methods

- Outside class:
using an object (must be in scope of course)
use dot notation
- Inside class definition:

variables and methods have scope local to the class and
can be used directly:
DO NOT USE DOT NOTATION

Class versus Object

- A class defines a data type

A data type characterizes how a set of entities is internally represented and which operations can be performed on it

A class is often described as a blueprint or template for creating objects. A class defines the (abstract) characteristics that objects share: the properties and the methods.

- Objects are the actual data variables that have the class as data type

Objects are instances of a class. All objects created from one class have the same property names and methods, but the variables of each can contain different data

Example V.2

Writing Complex Programs

Abstraction

- Decomposition: programming one thing at a time
 - into simpler parts
 - into parts that are natural for the problem
- Implementation hiding
offer a simple interface
- Abstraction levels
composition and inheritance

Implementation hiding

- In many languages access rights can be specified explicitly: public and private
- Often all properties and internal methods are made private; the public methods provide an **interface** to the class
- The interface hides the implementation. The implementation could change completely without a user of the class noticing...
- In Processing all properties and methods default to public

Abstraction levels

- **Composition** (**has-a** relations):
Class can have properties that are classes themselves

Example: Address class in Employee class

- **Inheritance** (**is-a** relations):
A class can **extend** an existing class and make it more specific

Example:

Animal: gender

|

Bird: beakColor, wingSpan

|

WaterBird: swimSpeed

Animal Farm