

Algorithmics 3

Blood, sweat and code

September 12, 2006

We whall see

- ▶ Object-oriented inheritance.
- ▶ Object-oriented polymorphism.
- ▶ Object-oriented encapsulation.
- ▶ Modules/packages.
- ▶ Interfaces and abstract classes.
- ▶ Exceptions.
- ▶ Graphic User Interfaces.
- ▶ File management.
- ▶ JavaDoc.

We whall see

- ▶ Object-oriented inheritance.
- ▶ Object-oriented polymorphism.
- ▶ Object-oriented encapsulation.
- ▶ Modules/packages.
- ▶ Interfaces and abstract classes.
- ▶ Exceptions.
- ▶ Graphic User Interfaces.
- ▶ File management.
- ▶ JavaDoc.
- ▶ If possible, Web Applets.
- ▶ If possible, Multimedia.

We whall see

- ▶ Object-oriented inheritance.
- ▶ Object-oriented polymorphism.
- ▶ Object-oriented encapsulation.
- ▶ Modules/packages.
- ▶ Interfaces and abstract classes.
- ▶ Exceptions.
- ▶ Graphic User Interfaces.
- ▶ File management.
- ▶ JavaDoc.
- ▶ If possible, Web Applets.
- ▶ If possible, Multimedia.
- ▶ If possible, who knows, a little bit of 3D programming.

Goals

By the end of the term, you should be able to

- ▶ From a problem, derive a set of Object-Oriented specifications.
- ▶ Understand and use Java's libraries.
- ▶ Build new libraries.
- ▶ Develop robust and well-documented software, with user interfaces, files ...

Goals

By the end of the term, you should be able to

- ▶ From a problem, derive a set of Object-Oriented specifications.
- ▶ Understand and use Java's libraries.
- ▶ Build new libraries.
- ▶ Develop robust and well-documented software, with user interfaces, files ...
- ▶ ... including a small video game (MineSweeper).

What you should already know (1)

Procedural imperative programming

- ▶ Defining procedures and functions.
- ▶ Calling procedures and functions.
- ▶ Local variables.
- ▶ Control structures (`if(...)...else...`, `case(...)`, `while(...)`.....)

If anything is missing, say so !

What you should already know (2)

Object-oriented programming

- ▶ Defining a class.
- ▶ Defining fields/attributes/object variables.
- ▶ Defining methods.
- ▶ Defining subclasses, inheriting.
- ▶ Constructors.
- ▶ Instanciation.

If anything is missing, say so !

What you may know

Data structures

- ▶ Unit (void).
- ▶ Primitive structures (integers, floating numbers, booleans, characters).
- ▶ Records.
- ▶ Arrays.
- ▶ Lists.
- ▶ Queues, Stacks.
- ▶ Sets, Maps.

Input/Output

- ▶ Character strings.
- ▶ Command-line output `System.out.println`
- ▶ Arguments input `public static void main(String[] args)`

If anything is missing, say so !

Evaluation

- Exercise sessions** During each session, we will single out a few students and ask questions related to the exercises of the previous session. This will receive a mark. Too bad for absents.
- End-of-term exam** At the end of the term, we will ask you to prove your mastery of Java and the Minesweeper developed in TD. In practice, you will be prompted to modify the source to add features, with limited time.

Understanding

- ▶ Nothing we teach you is beyond your intelligence. In the worst case, it may be ill-formulated or the lecturer may not understand some of your problems.
- ▶ Ask questions. As many as it takes. It will help you and everyone in the classroom, including the lecturer.
- ▶ Work with friends. Discuss what you understood and what you didn't. It will help everyone.
- ▶ Recapitulate what you have learnt at the end of every week. This will help you grow aware of your strengths and weaknesses.
- ▶ Not understanding is not a deadly sin. But you need to be aware of what you don't understand and you need to try and fix it.
- ▶ Sometimes, there's no use in spending hours in trying to understand something obscure. Take a break. Do something else. Go see a movie. Come back fresh and try again.
- ▶ Don't wait until the day before the exam. It's too late.
- ▶ Take notes.

Practicing

- ▶ Practice is necessary.
- ▶ Exercises are given for a reason. Try and understand it.
- ▶ Once model answers for an exercise have been given, make sure you understand them and make sure you are now able to solve them.
- ▶ Working in groups is good. Just make sure that everybody understands everything. Really.
- ▶ Why not join an open-source project ? It's a good opportunity to try some real-world problems, learn something useful, get to know interesting people and improve your resume.
- ▶ Practice your English. No matter what you do, you'll need it.
- ▶ Whenever you fail at something, take notes about what failed.

Looking for information

- ▶ Ask questions.
- ▶ You are in a research university. Chances are that some of your lecturers are world-level specialists of something. They can point you towards something
- ▶ If you're looking for additional information, look in the webography.
- ▶ Google is your friend. Just make sure that you keep a dose of skepticism in mind: not everything you read in the Internet is true.

Programming

- ▶ A program always tries to solve a problem. Yes, even games. Always be sure that you understand the problem before starting to write a program.
- ▶ A program which doesn't compile is not a program at all. Make sure your program compiles.
- ▶ When a program doesn't compile, read the error message. It will tell you whether you have a simple syntax error, a symbol error or a type error. Syntax errors are easy to fix. Symbol errors typically arise from typos, misunderstood documentations or missing libraries. Type errors usually indicate serious problems.
- ▶ Always prepare a set of tests to check that your program works.
- ▶ Always test your programs.
- ▶ Be sure you understand the error messages produced by the compiler.

Jobs (1)

Software architect/developer Understand the requirements of the client, understand the problem, derive specifications, implement the specifications as a robust program, deliver.

- ▶ Communication, rigor, organisation, sense of method, English.
- ▶ Programming skills (nowadays, most often Java or C#, with their libraries).
- ▶ Engineering degree/DESS/Master Pro.
- ▶ Internship in a company.

Jobs (1)

Software architect/developer Understand the requirements of the client, understand the problem, derive specifications, implement the specifications as a robust program, deliver.

- ▶ Communication, rigor, organisation, sense of method, English.
- ▶ Programming skills (nowadays, most often Java or C#, with their libraries).
- ▶ Engineering degree/DESS/Master Pro.
- ▶ Internship in a company.

Variants Software technician, project manager, video game developer.

Jobs (2)

Web developer Define a look and feel for a website, use or create the tools for managing the website.

- ▶ Communication, curiosity, tools usage.
- ▶ Some programming skills (most often Java Server Pages, Active Server Pages or PHP).
- ▶ Webpage skills.
- ▶ IUT/BTS.
- ▶ Internship in a company.

Jobs (2)

Web developer Define a look and feel for a website, use or create the tools for managing the website.

- ▶ Communication, curiosity, tools usage.
- ▶ Some programming skills (most often Java Server Pages, Active Server Pages or PHP).
- ▶ Webpage skills.
- ▶ IUT/BTS.
- ▶ Internship in a company.

Variants Multimedia developer.

Jobs (3)

System administrator Set up and manage the network of a company, install whichever software or hardware components are necessary, tighten security, find and kick intruder's ass. . .

- ▶ Adaptation, hacker's mindframe.
- ▶ English.
- ▶ Knowledge of sources of information.
- ▶ Extensive Operating System skills (most often either Windows or Linux).
- ▶ Engineering degree/DESS/Master Pro.

Jobs(4)

Researcher Invent the tools used by everyone else to do their job, or the theory someone will need to invent these tools.

- ▶ Adaptation, curiosity, good morale.
- ▶ English.
- ▶ Mathematics and computer science skills.
- ▶ Master Pro/Master Research + PhD.

The teacher

Name David Teller

Status Lecturer/Researcher (Maître de Conférences)

e-Mail David.Teller@univ-orleans.fr

Office To do !

???

.

Web To do !

???

.

Office hours Monday afternoons.

Specialties Tri-classed Programming Languages 8/Open-source
6/Software security 2.

A few more things to know (2)

Name Jérémy Briffaut

Status PhD Teaching Assistant (Thésard Moniteur)

e-Mail Jeremy.Briffaut@ensi-bourges.fr

Office ENSIB CRI-11.

Office hours All week long.

Web <http://www.synoptick.com>.

Specialties Software security.

Webography

- On-line Java Tutorials** To learn quickly some aspect of Java <http://java.sun.com/docs/books/tutorial/index.html>.
- Java's Libraries** To check in details how some aspect of Java works <http://java.sun.com/j2se/1.5.0/docs/index.html>.
- Java Language Specifications** To look smart and know what most people don't know about Java <http://java.sun.com/docs/books/jls/>.
- Passeport Informatique** A good website for whoever wishes to work in technologies of information and communication <http://www.passinformatique.com/>

Any questions ?

If you have any question, now is the time to ask.

Any questions ?

If you have any question, now is the time to ask.
After this, we start.

List of courses

- What we will do
- What you should already know
- What you should do
- What will happen to you
- About us

Object-oriented design

- From Poetry to Java
- From Batman to Java
- Conclusions

Thus spoke the teacher

Let's start with a specification.

*"Absence makes the
heart grow fonder.
And too much makes it
wander."*

(English proverb)

*"De l'absence et le cœur
se fait plus affectueux.
Trop d'absence et le
cœur s'enfuit."*

(English proverb)

Thus spoke the teacher

Let's start with a specification.

*"Absence makes the
heart grow fonder.
And too much makes it
wander."*

(English proverb)

*"De l'absence et le cœur
se fait plus affectueux.
Trop d'absence et le
cœur s'enfuit."*

(English proverb)

What entities do we have ?

Thus spoke the teacher

Let's start with a specification.

*"Absence makes the heart grow fonder.
And too much makes it wander."*

(English proverb)

*"De l'absence et le cœur se fait plus affectueux.
Trop d'absence et le cœur s'enfuit."*

(English proverb)

What entities do we have ?

- ▶ absence / too much absence
- ▶ heart
- ▶ fondness / more fondness
- ▶ wandering.

Thus spoke the teacher (2)

*“Absence makes the
heart grow fonder.
And too much makes it
wander.”*

- ▶ absence / too much absence
- ▶ heart
- ▶ fondness / more fondness
- ▶ wandering.

What is happening ?

Thus spoke the teacher (2)

“Absence makes the heart grow fonder. And too much makes it wander.”

- ▶ absence / too much absence
- ▶ heart
- ▶ fondness / more fondness
- ▶ wandering.

What is happening ?

- ▶ growing fonder (the heart)
- ▶ wandering (the heart).

Thus spoke the teacher (3)

*“Absence makes the
heart grow fonder.
And too much makes it
wander.”*

Triggered actions

- ▶ growing fonder (the heart)
- ▶ wandering (the heart).

?

- ▶ absence / too much
absence
- ▶ heart.

Who is it happening to ?

Thus spoke the teacher (3)

*“Absence makes the
heart grow fonder.
And too much makes it
wander.”*

Triggered actions

- ▶ growing fonder (the heart)
- ▶ wandering (the heart).

?

- ▶ absence / too much
absence
- ▶ heart.

Who is it happening to ?

- ▶ the heart (growing fonder, wandering).

Thus spoke the teacher (4)

*“Absence makes the heart grow fonder.
And too much makes it wander.”*

Triggered actions

- ▶ growing fonder (the heart)
- ▶ wandering (the heart).

Object

- ▶ the heart

?

- ▶ absence / too much absence

What makes it happen ?

Thus spoke the teacher (4)

*“Absence makes the heart grow fonder.
And too much makes it wander.”*

Triggered actions

- ▶ growing fonder (the heart)
- ▶ wandering (the heart).

Object

- ▶ the heart

?

- ▶ absence / too much absence

What makes it happen ?

- ▶ absence
- ▶ too much absence.

Thus spoke the teacher (5)

*“Absence makes the heart grow fonder.
And too much makes it wander.”*

Objects

- ▶ the heart

Actions

- ▶ growing fonder (the heart)
- ▶ wandering (the heart).

Properties

- ▶ fondness (the heart).

Messages

- ▶ absence / too much absence (the heart).

Object-Oriented programming

Object-oriented programming is all about

- ▶ objects
- ▶ the state of these objects (i.e. their properties)
- ▶ what the objects can do
- ▶ what messages the objects can answer to.

Here's my heart

```
public class Heart
{
    //Properties
    private int fondness      = 0;
    //Actions
    protected void wander()
    {
        //TO DO: Define "wandering"
    }
    protected void growFonder()
    {
        fondness = fondness + 1;           //Change the state of this h
    }
    //Messages
    public void absence(int absenceLevel)
    {
        if (absenceLevel <= MAX_ABSENCE_LEVEL) //Not too much absence
            this.growFonder();
        else //Too much absence
            this.wander();
    }
}
```

Why Object-Oriented Programming ?

- ▶ Object-Oriented Programming is a set of rules enforcing specific methods on how to describe problems and solve them.
- ▶ Object-Oriented programming is supposed to permit easy modelisation of problems – hence to facilitate the design of programs answering these problems.
- ▶ Object-Oriented programming permits a good organization of complex programs.
- ▶ It's simply not possible to make huge programs without using Object Orientation (or a competing paradigm).

Why Object-Oriented Programming ?

- ▶ Object-Oriented Programming is a set of rules enforcing specific methods on how to describe problems and solve them.
- ▶ Object-Oriented programming is supposed to permit easy modelisation of problems – hence to facilitate the design of programs answering these problems.
- ▶ Object-Oriented programming permits a good organization of complex programs.
- ▶ It's simply not possible to make huge programs without using Object Orientation (or a competing paradigm).

Note Java is *not* fully object-oriented.

Why Object-Oriented Programming ?

- ▶ Object-Oriented Programming is a set of rules enforcing specific methods on how to describe problems and solve them.
- ▶ Object-Oriented programming is supposed to permit easy modelisation of problems – hence to facilitate the design of programs answering these problems.
- ▶ Object-Oriented programming permits a good organization of complex programs.
- ▶ It's simply not possible to make huge programs without using Object Orientation (or a competing paradigm).

Note Java is *not* fully object-oriented. But we're going to ignore that.

A more difficult specification

Let's build a simulator for flying bats:

- ▶ When two bats get too close to each other, they will inflect their course of 10 degrees each, so as to build distance between them.
- ▶ When a bat gets too far from the center of the flock, it will inflect its course of 5 degrees so as to get closer to the center.
- ▶ Lorsque deux chauve-souris s'approchent trop l'une de l'autre, elles vont modifier leur trajectoire de 10 degrés chacune, de manière à s'écarter l'une de l'autre.
- ▶ Lorsqu'une chauve-souris s'écarte trop du centre de la volée, elle va modifier sa trajectoire de 5 degrés de manière à se rapprocher du centre.

A more difficult specification

Let's build a simulator for flying bats:

- ▶ When two bats get too close to each other, they will inflect their course of 10 degrees each, so as to build distance between them.
- ▶ When a bat gets too far from the center of the flock, it will inflect its course of 5 degrees so as to get closer to the center.
- ▶ Lorsque deux chauve-souris s'approchent trop l'une de l'autre, elles vont modifier leur trajectoire de 10 degrés chacune, de manière à s'écarter l'une de l'autre.
- ▶ Lorsqu'une chauve-souris s'écarte trop du centre de la volée, elle va modifier sa trajectoire de 5 degrés de manière à se rapprocher du centre.

Note That's actually something you could do in your spare time. It looks fun. Or maybe we're going to do it in TD.

Invasion of the mutant vampire bats

- ▶ When two bats get too close to each other, they will inflect their course of 10 degrees each, so as to build distance between them.
- ▶ When a bat gets too far from the center of the flock, it will inflect its course of 5 degrees so as to get closer to the center.

Entities ?

Invasion of the mutant vampire bats

- ▶ When two bats get too close to each other, they will inflect their course of 10 degrees each, so as to build distance between them.
- ▶ When a bat gets too far from the center of the flock, it will inflect its course of 5 degrees so as to get closer to the center.

Entities

- ▶ bats
- ▶ distance between bats
- ▶ minimal distance
- ▶ maximal distance
- ▶ course – measured in degrees
- ▶ the flock of bats
- ▶ distance between a bat and the flock
- ▶ center of the flock.

Invasion of the mutant vampire bats

- ▶ When two bats get too close to each other, they will inflect their course of 10 degrees each, so as to build distance between them.
- ▶ When a bat gets too far from the center of the flock, it will inflect its course of 5 degrees so as to get closer to the center.

Invasion of the mutant vampire bats

- ▶ When two bats get too close to each other, they will inflect their course of 10 degrees each, so as to build distance between them.
- ▶ When a bat gets too far from the center of the flock, it will inflect its course of 5 degrees so as to get closer to the center.

Objects Bats, flocks.

Invasion of the mutant vampire bats

- ▶ When two bats get too close to each other, they will inflect their course of 10 degrees each, so as to build distance between them.
- ▶ When a bat gets too far from the center of the flock, it will inflect its course of 5 degrees so as to get closer to the center.

Objects Bats, flocks.

Properties of Bat Position, course.

Messages of Bat Another bat is too close. This bat is too far from the center.

Actions of Bat Inflect course (how many degrees ? which way ?)

Invasion of the mutant vampire bats

- ▶ When two bats get too close to each other, they will inflect their course of 10 degrees each, so as to build distance between them.
- ▶ When a bat gets too far from the center of the flock, it will inflect its course of 5 degrees so as to get closer to the center.

Objects Bats, flocks.

Properties of Bat Position, course.

Messages of Bat Another bat is too close. This bat is too far from the center.

Actions of Bat Inflect course (how many degrees ? which way ?)

Properties of Flock All the bats. Position of the center.

Messages of Flock None.

Actions of Flock None.

Flocks

```
public class Flock
{
    //Properties
    private Bat[] allTheBats;
    private Point positionOfCenter; //Note: that's not the only way of doing this
    //Messages
    //Actions
}
```

Flocks

```
public class Flock
{
    // Properties
    private Bat[]    allTheBats;
    private Point    positionOfCenter; // Note: that's not the only way of doing this
    // Messages
    // Actions
}
```

Note that Flock *depends on* Bat.

Bat

```
public class Bat
{
    // Properties
    private Point    position;
    private Direction course;
    // Messages
    public void batTooClose(Point positionOfTheOtherBat)
    {
        this.infectCourse(10, awayFrom(positionOfTheOtherBat));
        //TO DO: define "awayFrom"
    }
    public void centerTooFar(Point positionOfTheFlock)
    {
        this.infectCourse(5, towards(positionOfTheFlock));
        //TO DO: define "towards"
    }
    // Actions
    protected void infectCourse(int degreesOfInflection, Direction inDirection)
    {
        //TO DO: Define "infectCourse"
    }
}
```

Bottom line

1. Start from the specifications.
2. Find out what *objects* are involved.
3. Determine what *properties* these objects have.
4. Determine what *messages* these objects can receive.
5. Determine what *actions* these objects can undertake.

Bottom line

1. Start from the specifications.
2. Find out what *objects* are involved.
3. Determine what *properties* these objects have.
4. Determine what *messages* these objects can receive.
5. Determine what *actions* these objects can undertake.

Next time, we will discuss relation *between* objects.

Any questions ?