# Course description

Lecture 1

# The course is about

- Object-oriented approach to programming
- Fundamentals of system modeling
- Java programming skills

# Course audience

Who is it for?

- students with varying levels of programming experience -- including NONE!
- anyone who wants to master craft of program design and implementation

# Why Java? Methodological

- Fully Object-Oriented language
- Simple clean small type system

# Java syntax is simple

```
int size = 27;

String name = “Fido”;

Dog myDog
    = new Dog(name, size);

int x = size - 5;

if (x < 15)
    myDog.bark(8);

while (x > 3)
{
    myDog.play();
}

int [ ] numList = {2,4,6,8};
```

```
System.out.print(“Hello”);

System.out.print(“Dog: “ +
                        name);
String num = “8”;
int z = Integer.parseInt(num);

try
{
    readTheFile(“myFile.txt”);
}
catch(Exception ex)
{
    System.out.print
            (“File not found.”);
}
```

# Why Java? Methodological

- Fully Object-Oriented language
- Simple clean small type system
- Platform-independent: write once, run everywhere (in principle)
- Java does a lot more bookkeeping and resource management for you: concentrate on the task, not on the computer organization

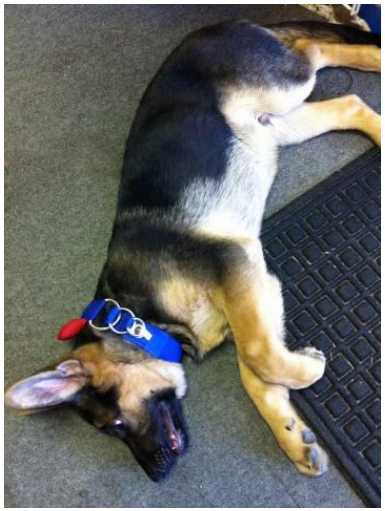
# Monday morning at Bob's


Delay the coffee 20 minutes

Hold the toast

Get the paper, no walk

Call the meeting: we are late



# All devices communicate in Java

- ***His toast is toasted.***
- ***His coffee steams.***
- ***His paper awaits.***

Just another wonderful morning in ***The Java-Enabled House*****.**

# Why Java? Methodological

- Fully Object-Oriented language
- Simple clean small type system
- Platform-independent: write once, run everywhere (in principle)
- Java does a lot more bookkeeping and resource management for you: concentrate on the task, not on the computer organization

# Java drawbacks

Slow: due to

- Garbage collector
- Compilation from byte code to JIT – Just In Time – machine code

# Why Java? Pragmatic

- Large projects with short schedule: divide work into components
- Long-lived, reliable, modifiable software

- It is the most prevalent language for the Web, and one of the most prevalent languages in industry today (others include C, C++, Python, Ruby…etc.)

http://langpop.com/

(Programming Language Popularity.htm)

# Course distinctive features

- Building small building blocks and combining them into a more complex system
- Using a high-level language as a black-box – less technical details of what is actually happening in the machine
- Solving problems in your own way – develop creativity

# Textbook

**Introduction to Programming and Object-Oriented Design Using Java** by Jaime Nino and Frederick A. Hosch, 2008, ISBN-10: 0470128712

# Optional, used for this course

- Free Electronic Book: **Thinking in Java** by Bruce Eckel
- **Head First Object-Oriented Analysis and Design** by Brett D. McLaughlin, Gary Pollice and Dave West
- **Object-Oriented Programming in Java: A Graphical Approach**, 2005, by Sanders, van Dam ISBN-10: 0321245741

# Declarative and imperative knowledge

- Declarative knowledge: Nanaimo is on Vancouver Island
  - Explicit knowledge: know-what (facts), know-why (science)
- Imperative knowledge: can only be revealed through practice in a particular context: riding a bicycle
  - Implicit knowledge: know-how

# Course assessment

This is a practical lab-based hands-on course

- 10 programming assignments – total 50 points
- 10 home quizzes (test understanding of concepts) – total 30 points
- Final project – total 20 points

# Course mechanics

Early and Late Hand-In Policy

- submit programs 2 days early for extra credit
- penalty for programs handed in up to 2 days late

Keys to success

- start early, work steadily, don’t fall behind
- you can’t cram, unlike other courses
- exponential growth of program size throughout the semester

# Collaboration

Programming is learnt best when student is **solving problems on his own**, with adequate **help from the instructor**

- The course grade is entirely based on programs & home works, all your own work. Only collaboration that is properly documented is allowed, while working on the final project.
- Plagiarism detection with MOSS (Measure of Software Similarity)

http://theory.stanford.edu/~aiken/moss/

- Punishments: zero grade, penalty grade, suspension (no jail time).

# From: **VIU Academic code of conduct**

Plagiarism is the unacknowledged use of someone else's words, ideas, or data regardless of source. When a student submits work for credit that includes the words, ideas or data of others, the source of that information must be acknowledged through complete, accurate, and specific references. By placing their names on work submitted for credit, students certify the originality of all work not otherwise identified by appropriate acknowledgments.

# Lecture objectives

To be able to:

- Explain what is Object-Oriented Programming (OOP)
- Understand the concept of abstraction
- Understand the concept of encapsulation

# Abstraction

- Each piece of code is an abstraction of a real-life object or process
- Abstraction contains only important features

# History: the progress of abstraction I

- The 1950s – Machine code is common. Assembly language abstracts an underlying computing machine
- The 1960s - “Imperative” languages (FORTRAN, BASIC, C) – built as an abstraction level upon Assembly. Their primary abstraction still requires you to think in terms of the structure of the computer rather than the structure of the problem you are trying to solve.

# History: the progress of abstraction II

- The 1970s - The alternative to modeling the machine is to model the problem you're trying to solve. Early languages such as LISP and APL chose particular views of the world ("All problems are ultimately lists" or "All problems are algorithmic," respectively). PROLOG casts all problems into chains of decisions.

# History: the progress of abstraction III

- The 1980s – “modular” languages (Modula-2, ADA) – can work on each part separately, precursors of modern OOP

# The structured programming paradigm: daily activities example

Split this list into three blocks of related activities and give each block a heading

- Get out of bed
- Eat breakfast
- Park the car
- Get dressed
- Get the car out of the garage
- Drive to work
- Find out what your boss wants you to do today
- Feedback to the boss on today's results.
- Do what the boss wants you to do

# Daily activities modules

**Get up**

- Get out of bed
- Get dressed
- Eat breakfast

**Go to work**

- Get the car out of the garage
- Drive to work
- Park the car

**Do your job**

- Find out what your boss wants you to do today
- Feedback to the boss on today's results.
- Do what the boss wants you to do

# We can work on each part separately

Improve instructions in **go to work** module:

- Listen to the local traffic and weather report
- Decide whether to go by bus or by car
- If going by car, get the car and drive to work.
- Else walk to the bus station and catch the bus

Other modules are not affected

# History: the progress of abstraction III

- The 1980s – “modular” languages (Modula-2, ADA) – can work on each part separately, precursors of modern OOP
- The 1990s – Object-Oriented paradigm and component-based programming. Wide use of OOP languages (Simula '67 and Smalltalk '72 )

- From 2000 – OOP is a standard programming paradigm for developing complex systems

# OOP – flexible abstraction of the problem space

- OOP provides tools for the programmer to represent elements in the problem space. This representation is general enough that the programmer is not constrained to any particular type of problem.
- We refer to the elements in the problem space and their representations in the solution space as “objects.”
- The idea is that the program is allowed to adapt itself to the lingo of the problem by adding **new types of objects**, so when you read the code describing the solution, you’re reading words that also express the problem.

# Object-Oriented Programming paradigm (by Alan Kay)

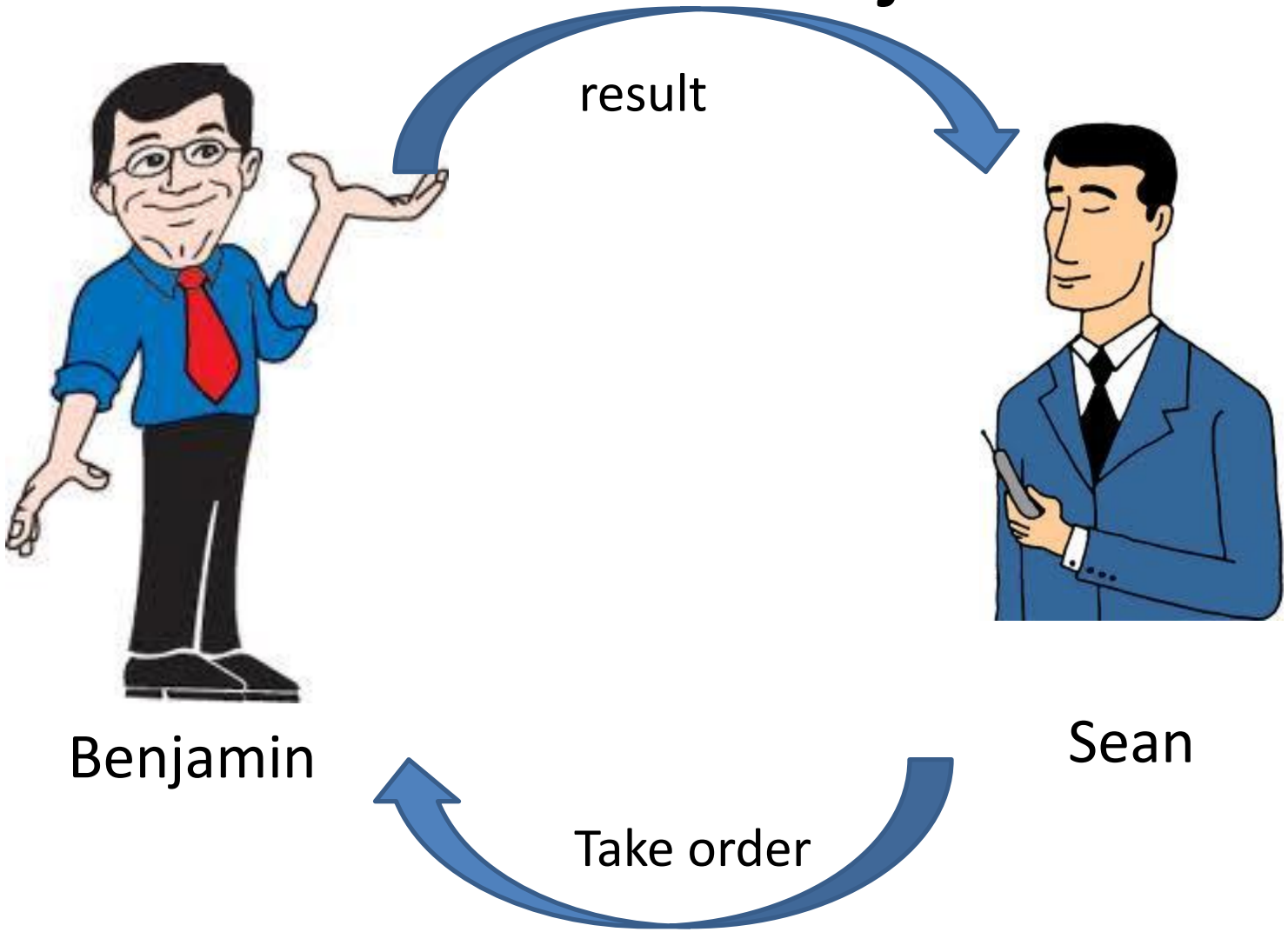
- Everything is an object.
- A program is a bunch of objects telling each other what to do by sending messages.
- Each object has its own memory made up of other objects.
- Every object has a type.
- All objects of a particular type can receive the same messages.

Alan Kay, received ACM's Turing Award, the "Nobel Prize of Computing," in 2003 for Smalltalk, the first complete dynamic OOPL

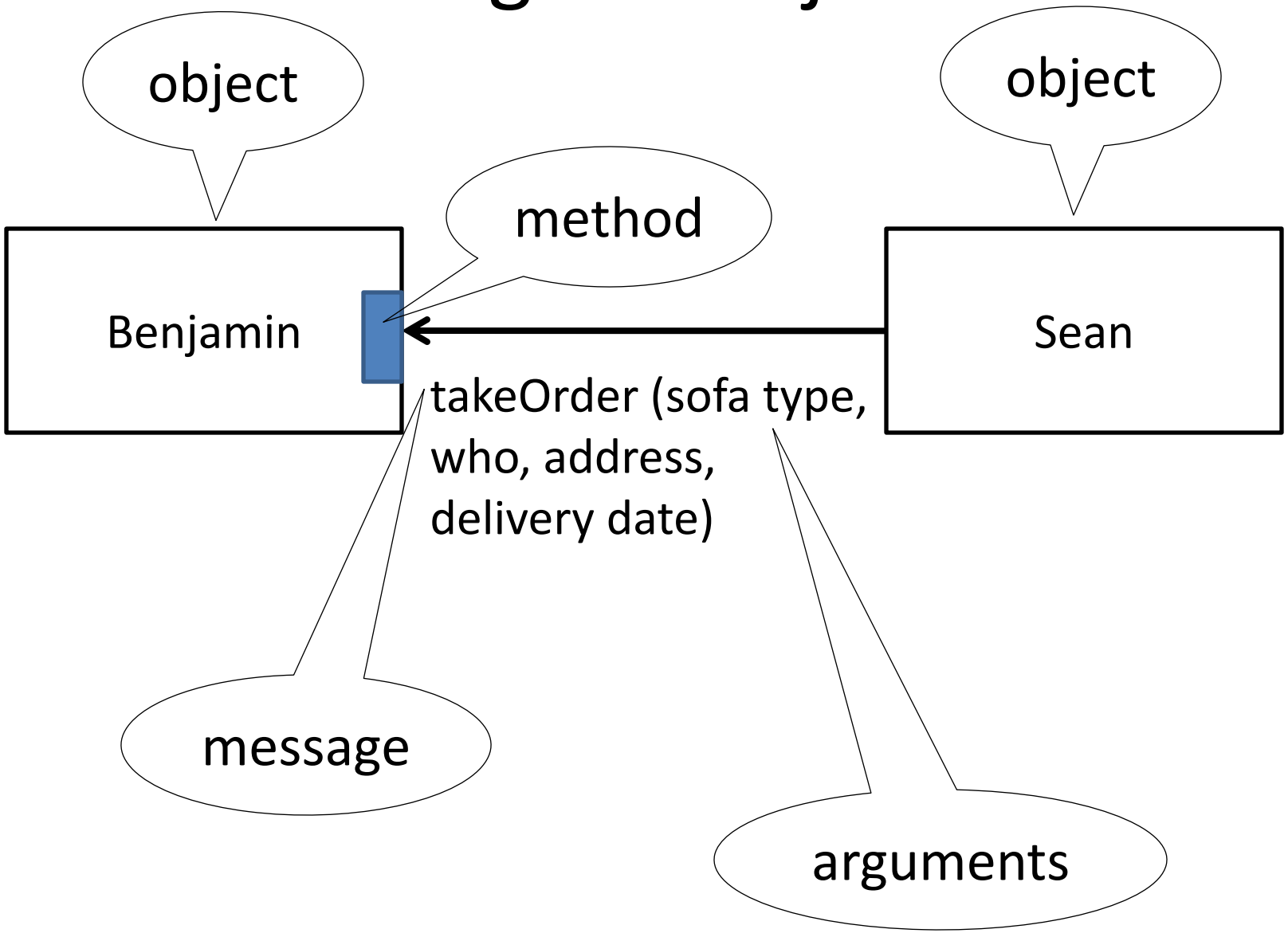
# OOP benefits

- We can create and manipulate objects independently of each other:
  - **change** and improve without affecting the working code
- By concentrating on one object at a time we can **simulate complex systems**
- We can **reuse** the same components in multiple scenarios

# Real-world objects



# Program objects



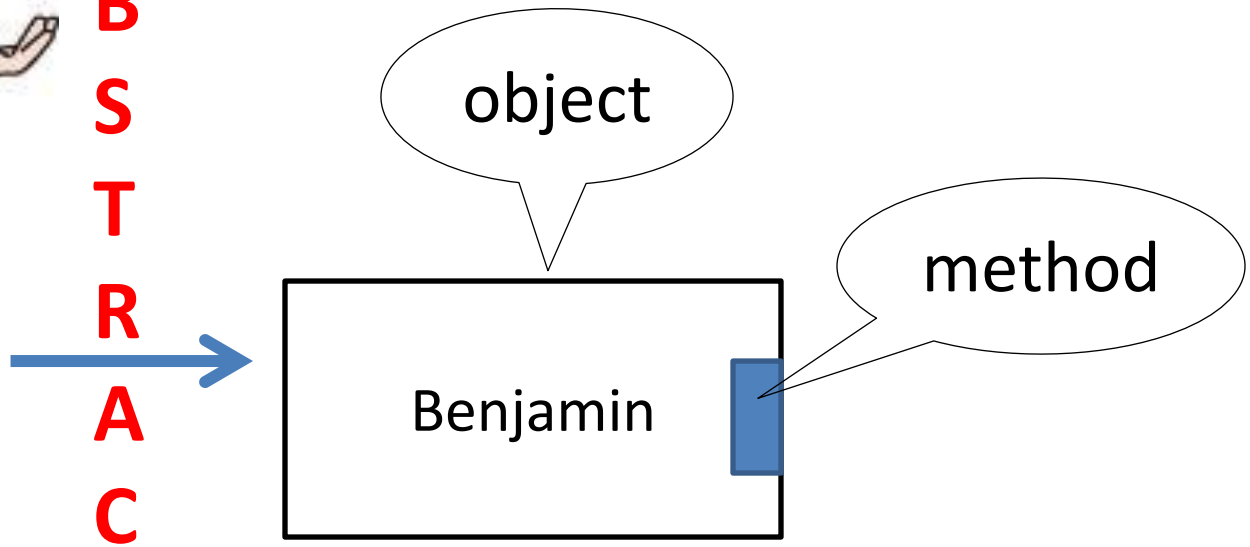
# OOP principles (just a beginning)

- Abstraction
- Encapsulation

# Abstraction

Benjamin

ABSTRACTION




Only important features

Importance is task-oriented

# Encapsulation

- The implementation is hidden from the outside world
- Only method signature is outward-facing and is accessible from outside. This is called *object interface*

# Summary

- We model real world objects by **abstracting** selected properties and actions of these objects, ignoring details.
- The **Object-oriented program** is a system of collaborating objects. They collaborate by sending messages.
- The outside objects do not know how object A does its thing. Object A **encapsulates** its methods, and shows only method signatures – **interface**.


We're going to Objectville! We're leaving this dusty ol' procedural town for good. I'll send you a postcard.