Object-Oriented Programming

CSE703029

Faculty of Computer Science

Phenikaa University

Lecture 1: Introduction of OOP

Slides adapted from Steven Roehrig

Expected Background

- “A one-semester college course in programming.”
- I assume you can write a program in some language, understand variables, control structures, functions/subroutines.
- If in doubt, let’s talk.

Course Outline

- Week 1: Background, basics of O-O, first Java program, programming environments
- Week 2: Raw materials: types, variables, operators, program control
- Week 3: Classes: declarations, constructors, cleanup & garbage collection
- Week 4: Packages, access specifiers, finals, class loading

Course Outline (cont.)

- Week 5: Polymorphism, abstract classes, design patterns
- Week 6: Interfaces & extends, inner classes, callbacks via inner classes
- Week 7: Applets, Applications, Swing
- Week 8: Graphics and Multimedia
- Week 9: Arrays, container classes, iterators

Course Outline (cont.)

- Week 10: Exception handling, threads
- Week 11: Java I/O, networking
- Week 12: JDBC and object persistency

Administrative Details (cont.)

- Required Text: Horstmann & Cornell, “Core Java, Volume 1 - Fundamentals,” Sun Microsystems Press
- Additional Online Texts:
  - The Java Tutorial
    http://java.sun.com/docs/books/tutorial/
  - Thinking in Java by B**ruce Eckel**

https://nglthu.github.io/Books/java/BruceEckel_Thinking_in_Java_4th_Edition.pdf

Administrative Details (cont.)

- 20 weeks
- Two and/or three consecutive 180-minute lectures per week. Breaks in between the lectures.
- Course notes in PowerPoint or HTML.
- Course notes available (usually) Wednesday before class.
- Send course questions to Canvas's Discussion Group, personal questions to me.

Administrative Details (cont.)

- Midterm and final exams
- Attendance: 10%
- Midterm exams (LTs+TH) 40%
- Final exam: 50%

Administrative Details (cont.)

- Everything is attached to the syllabus. Don't look for assignments, etc. on Blackboard. Look at the syllabus!
- Homework usually weekly.
- Submission instructions with each assignment, usually printed listings and a zip file.
- Printing slides? Three to a page, at least. Save a tree! Remove the PowerPoint background before printing. Save toner!

Administrative Details (cont.)

- Attendance is not required, but…10% &
- …you are responsible for everything said in class.
- I encourage you to ask questions in class. Don’t guess, ask a question!

My Policy on Cheating

- Cheating means “submitting, without proper attribution and citation, any computer code that is directly traceable to the computer code written by another person.”
- A failing *course* grade for *any* cheating. Expulsion is also possible.
- This doesn’t help your job prospects.

My Policy on Cheating

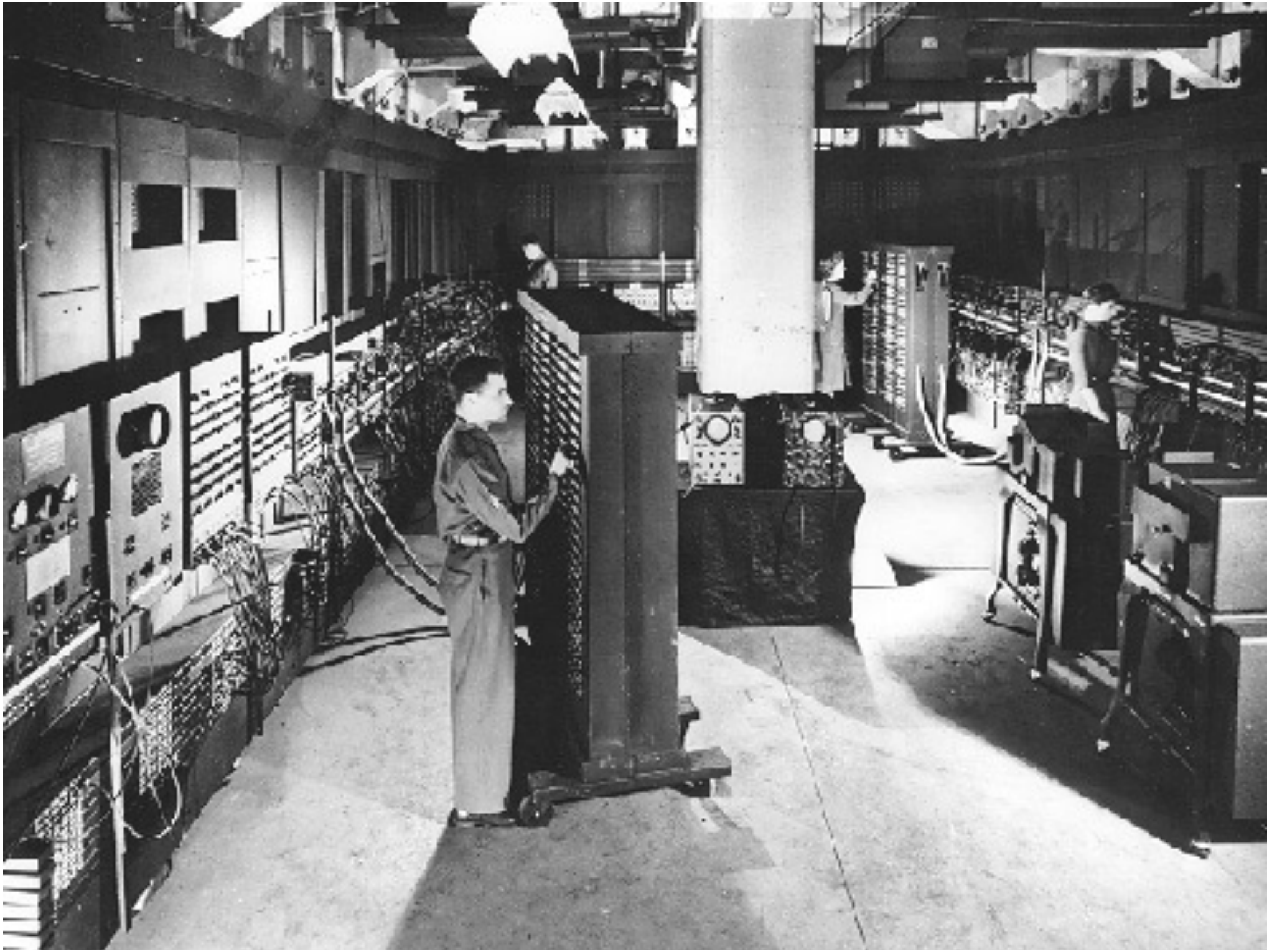
- You may discuss homework problems with classmates (teamwork), but you contribute your thoughts and work.
- You can use ideas from the literature (with proper citation).
- You can use anything from the textbook/notes.
- The code you submit **must be written completely by you**.

Even More On Cheating

- “In addition to any penalties imposed by the instructor, including failing the course, all cheating and plagiarism infractions will be reported in writing to the Associate Dean for the program, the Associate Dean of Faculty, the Dean of Student Affairs, and the Dean. They will review and determine if expulsion should be recommended. The report will become part of the student’s permanent record.”

Course Etiquette

- Etiquette is “conduct in polite society”
- No cell phones
- No random comings and goings
- If you are sleepy, go home
- If you want to read email or surf the Web, please do it elsewhere



Programming Language Evolution

- Machine language
- Assembler
- “3rd generation” (COBOL, FORTRAN, C)
- Specialized (Lisp, Prolog, APL)
- “4th generation” (SQL, spreadsheets, Mathematica)
- “5th generation” (example, anyone?)

Why So Many Languages?

- Bring the language “closer” to the problem.
- But 4GLs are typically focused on specialized domains (e.g., relational databases).
- We want a language that is general purpose, yet can easily be “tailored” to any domain.

Object-Oriented Languages

- Smalltalk, C++, Java, etc…
- You can make any kind of objects you want
- How different from procedural languages?
  - No different at all: Every (reasonable) language is "Turing complete"
  - Very different: Make expression easier, less error-prone

O-O Languages (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, and is made up of other objects.
- Every object has a type (class).
- All objects of the same type can receive the same messages.

Making Java Work

- It's “easy as pie” to write procedural code in Java.
- It takes some discipline (an attitude) to think O-O.
- It's worthwhile to do it.

Objects

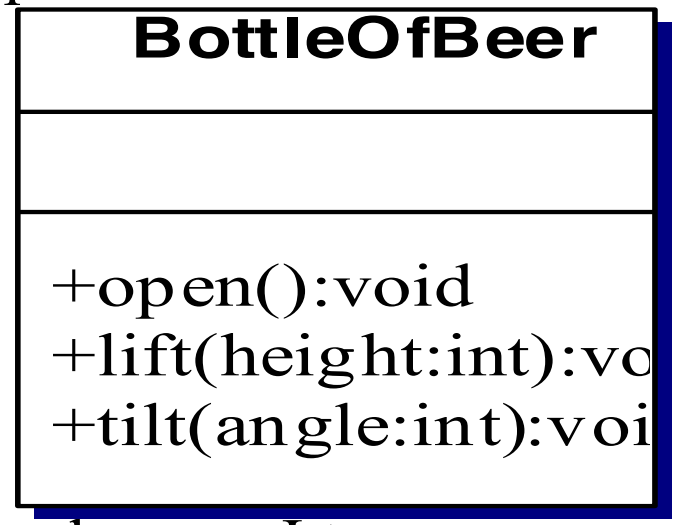
- An object has an interface, determined by its class.
- A class is an *abstract data type*, or *user-defined type*.
- Designing a class means defining its interface.

Built-In Types

- Think of an **int**…
  - What is its interface?
  - How do you "send it messages"?
  - How do you make one?
  - Where does it go when you're done with it?
- In solving a computational problem, the goal is to
  - Dream up useful classes, and
  - Endow them with appropriate characteristics.

Example

- Suppose I've defined this class in Java:



- To make one, I type
  BottleOfBeer myBeer = new BottleOfBeer();
- If I want myBeer opened, I say
  myBeer.open();

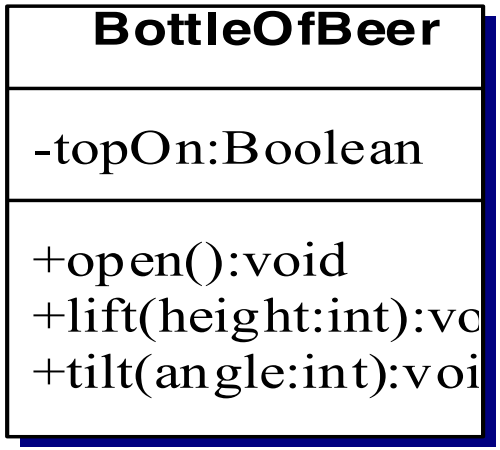
But Why Not Just…

BottleOfBeer myBeer, yourBeer, ourBeers;

- This is legal, but just makes a "reference variable" named **myBeer [to Me], yourBeer [to You], ourBeers [to Us] and ALL is Bottle Of Beer**
- This variable can refer to *any* **BottleOfBeer Object**, but currently refers to nothing
- The operator **new** actually causes an object to be created, so we tell it what kind we want

Designers Design, Users Use

- The interface is the critical part, but the details (implementation) are important too.



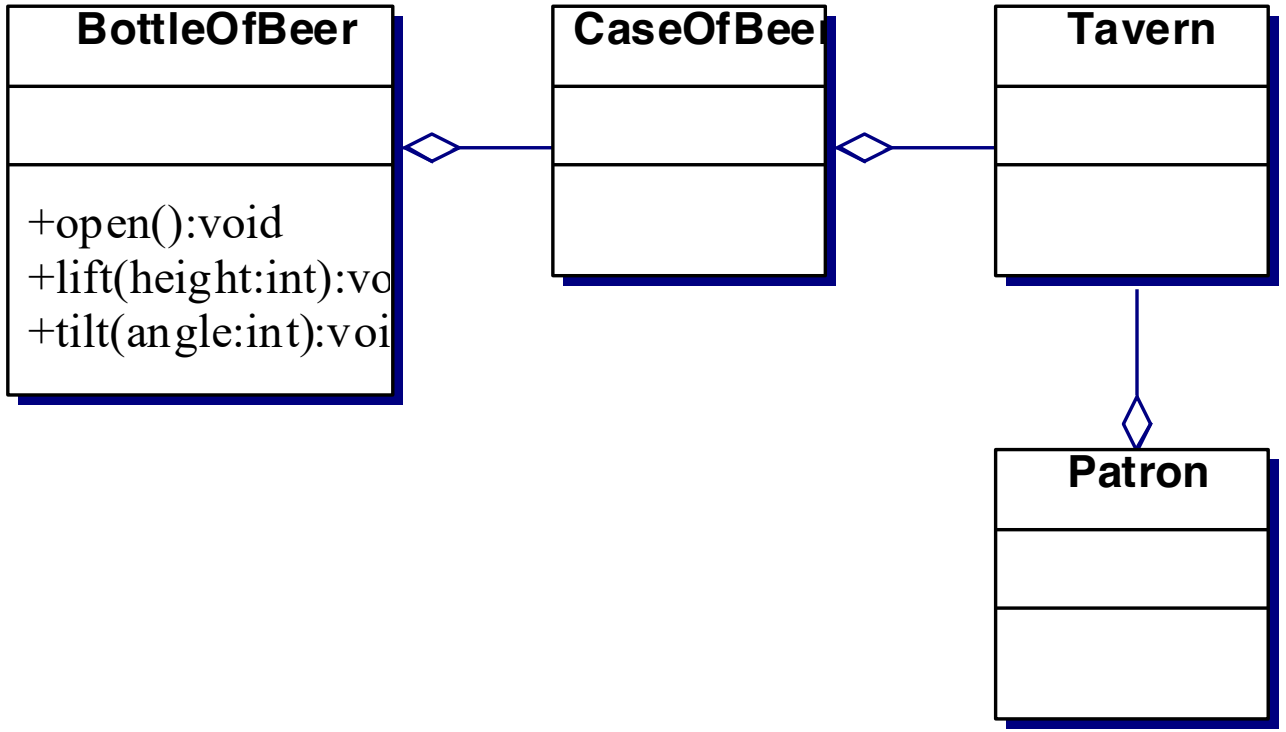
- Users use the interface (the “public part”); the implementation is hidden by “access control”.

Objects vs. Procedural Libraries

- C libraries are like this, sort of:
  - The library designer invents a useful struct.
  - Then she provides some useful functions for it.
  - The user creates an instance of the struct, then applies library functions to it.
- One big difference is that *anyone* can change any part of the struct. Booo, hsss!
- Another difference is in initialization.

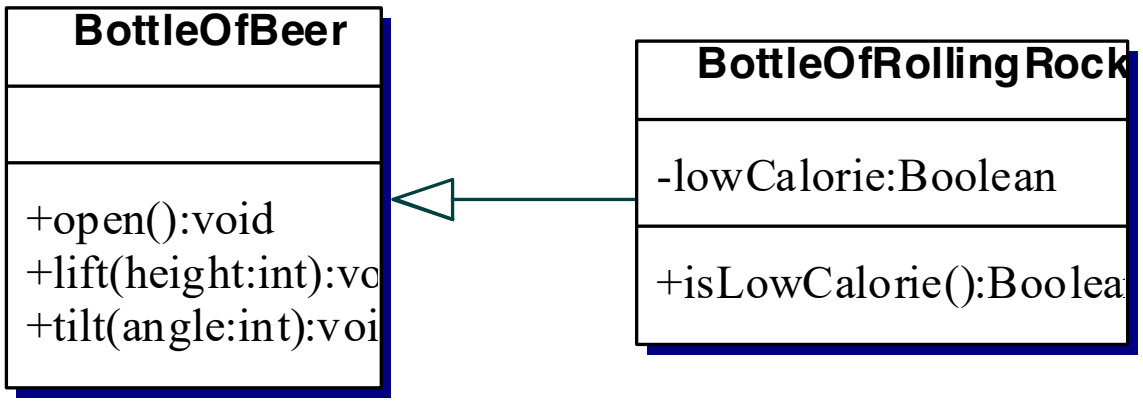
Two Ways of Reusing Classes

- Composition: One class has another as a part (indicated by the diamond “aggregation” symbol).



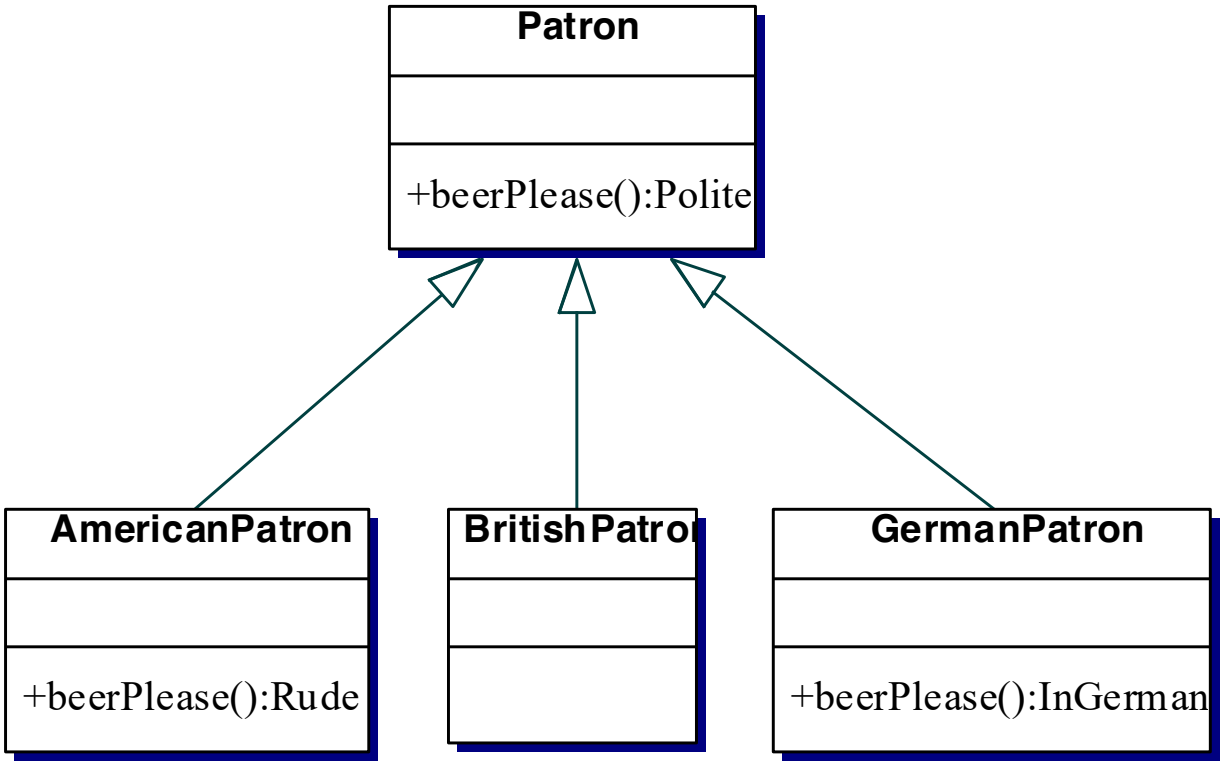
Two Ways of Reusing Classes

- Inheritance: One class is a specialized version of another (indicated by the triangle "inheritance" symbol).



Polymorphism

- Different subclasses respond to the same message, possibly with different actions.



Some Java Code

```
Patron p1 = new Patron();
Patron p2 = new YankPatron();
Patron p3 = new BritPatron();
Patron p4 = new GermanPatron();
p1.BeerPlease()      // polite request
p2. BeerPlease()     // rude request
p3.BeerPlease()      // polite request
p4.BeerPlease()      // request in German (but polite)
```

- This is a bit of a trick: it requires late binding of the function call.

Creating Objects

- We usually assume this is free; with built-in types like **int** or **char**, we just say

  int i;

  char c;

- With user-defined types (the ones we make), we need to be explicit about what we want:
  - constructor function
- This is a very important issue!

Destroying Objects

- If an object goes “out of scope,” it can no longer be used (its name is no longer known).
- In C++, we might need to write an explicit function to free memory allocated to an object.
- Java uses references and “garbage collection”.

Example of Object Scope

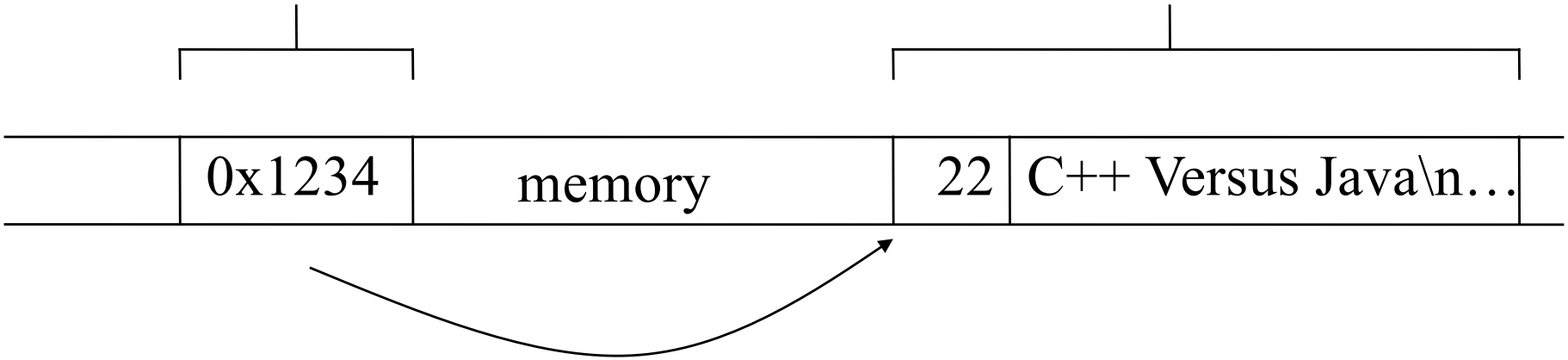
```
public String getTitle(int lectureNumber) {
    LectureNotes lect;
    lect = syllabus.getLecture(lectureNumber);
    String s = lect.getLine(1);
    return s;
}
```

- What happens to **lect**?
- The **LectureNotes** object still exists, but the reference **lect** disappears (it's out of scope after return) [**internal variable Only**]
- Eventually, the garbage collector removes the actual **LectureNotes** object.

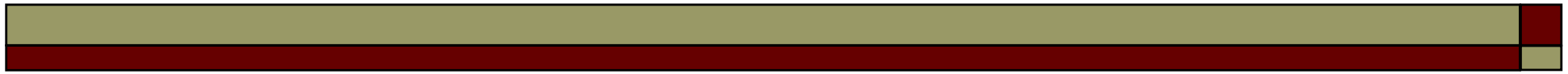
A Simple Model of Memory

The reference variable **lect** holds a memory location

The **LectureNotes** object is at that location



When you "speak of" the variable **lect**, you are "referring to" the actual **LectureNotes** object. When **lect** goes **out of scope** it is automatically **destroyed**. The **LectureNotes** object lives on, but nobody can use it…unless there are other references to it



Java's Use of Memory

- Stack
- Heap
- Static variables
- Constants
- Non-RAM storage

Java's Primitive Types

Type	Size	Wrapper type
boolean	-	Boolean
char	16-bit	Character
byte	8-bit	Byte
short	16-bit	Short
int	32-bit	Integer
long	64-bit	Long
float	32-bit	Float
double	64-bit	Double
void	-	Void

Wrapper Types

- Variables of **primitive types [**lower case**]**are "automatic", i.e., they are stored on the stack.
- They are automatically deleted when they go out of scope.
- What if you want an object holding a primitive type? Example:

char c = 'x'; Primitive type

Character C = new Character('x');

Really *Big* Numbers [Wrapper Types]

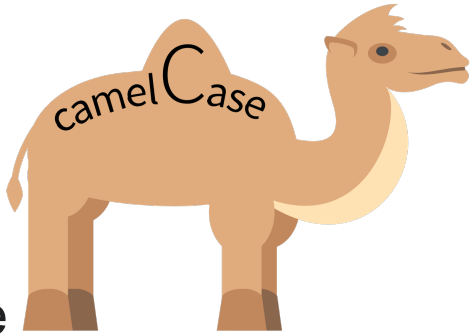
- **BigInteger**, **BigDecimal [Not Primitive Types]**
- These are arbitrary precision, as big as they need to be.
- You can't use the usual operators (+-*/) since they are objects. But there are methods (functions) to do these things.

Creating New Types

```
class MyNewType {
    // definition here
}
```

- Now it's legal to say

```
MyNewType m = new MyNewType();
// m is Object = Instance of Class
```



camelCase or CamelCase

Class Members

- Fields (member variables, data members)
- Methods (member functions)

```
class MyClass {
    int a;
    YourClass b;
    float memberFunction(int x, float f) {
        return 0;
    }
}
```

Let's Write Something

```
// Our first program. File: HelloDate.java
// Note that the file name is exactly the same
// as the class name, including capitalization.
import java.util.*;

public class HelloDate {
    public static void main(String[] args) {
        System.out.println("Hello, it is ");
        System.out.println(new Date());
    }
}
```

The Major Issues

- Editing
  - Use any text editor you like (*not* a word processor!); save as **HelloDate.java**
- Compiling
  - From a DOS or UNIX command line, type

    > javac HelloDate.java
  - This should produce the file **HelloDate.class**
- Running
  - Again from the command prompt, type

    > java HelloDate

Getting the Java Documentation

- Point your browser to

http://java.sun.com/docs/index.html

- Download version 1.4.2 of the J2SE API
- Bookmark the file so you can always get to it easily
- Or, just bookmark the index page on Sun's Java Web site