COIS 2240

Lecture 2

Requirements Elicitation

Req. Spec. & Analysis

Architecture Design

Detailed Design

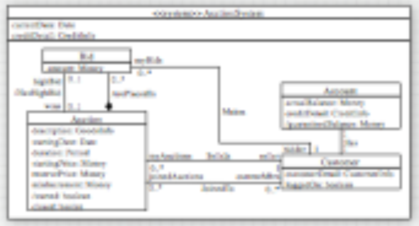
Implementation

Elevator Control System
Take Elevator
Call Elevator
Ride Elevator
Elevator Arrival
User
Exterior FloorButton
Interior FloorButton
Motor
Door
Floor Sensor

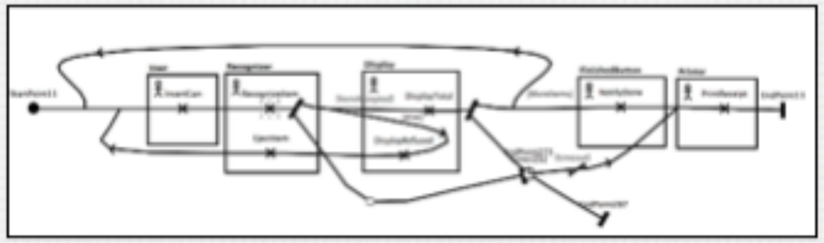
Use Case Model

Cost
Security
Authentication
Fingerprint
Password
Cardkey

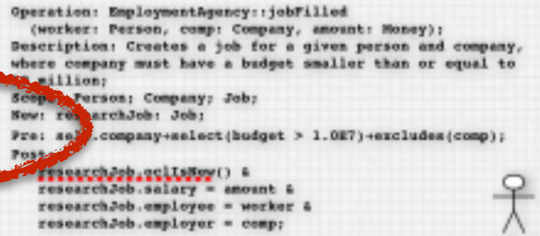
Goal Model



Concept Model

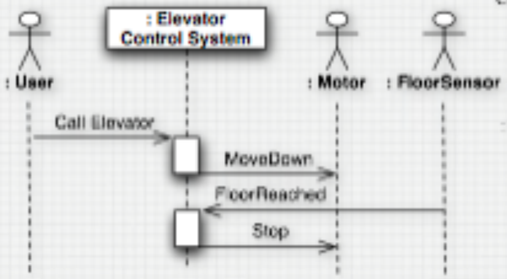


Use Case Maps

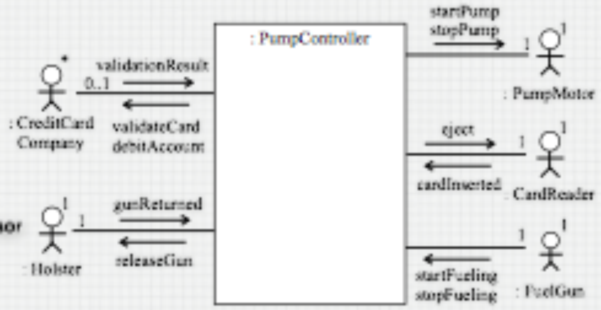


```
Operation: EmploymentAgency::jobFilled
  (worker: Person, comp: Company, amount: Money);
Description: Creates a job for a given person and company,
where company must have a budget smaller than or equal to
[illegible] million;
Scope: Person; Company; Job;
New: researchJob: Job;
Pre: self.company+select(budget > 1.0E7)+excludes(comp);
Post:
   researchJob.oclIsNew() &
   researchJob.salary = amount &
   researchJob.employee = worker &
   researchJob.employer = comp;
```

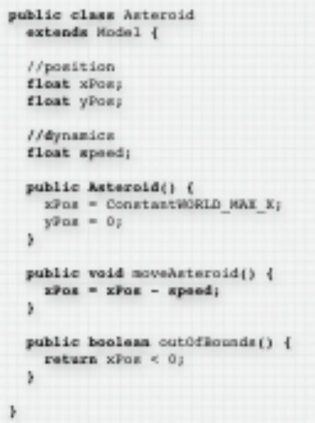
Operation Model



Protocol Model

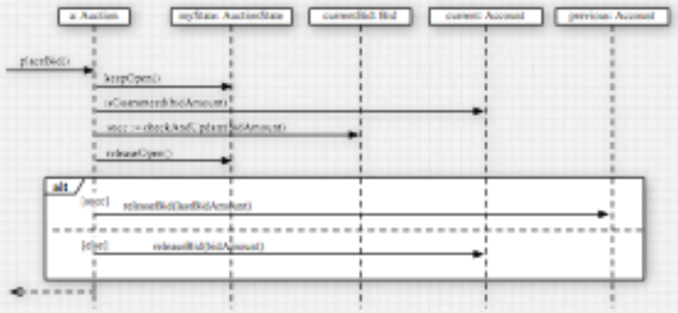


Environment Model



```
public class Asteroid
  extends Model {

  //position
  float xPos;
  float yPos;

  //dynamics
  float speed;

  public Asteroid() {
    xPos = ConstantWORLD_MAX_X;
    yPos = 0;
  }

  public void moveAsteroid() {
    xPos = xPos - speed;
  }

  public boolean outOfBounds() {
    return xPos < 0;
  }

}
```

Java Code



Interaction Model

Design Class Model

© Lethbridge/Laganière 2005

What is UML?

- UML (Unified Modeling Language)
  - Nonproprietary standard for modeling software systems, OMG
  - Information at the OMG portal http://www.uml.org/
- Commercial tools: Rational (IBM),Together (Borland), Visual Architect (business processes, BCD)
- Open Source tools: ArgoUML, StarUML, Umbrello,
- Online tools, look at this list: http://modeling-languages.com/web-based-modeling-tools/
- We will use umple umple.org

UML: First Pass

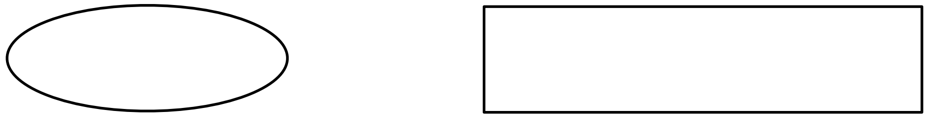
- You can model 80% of most problems by using about 20 % UML
- We teach you those 20%

UML First Pass

- Class diagrams
  - Describe the static structure of the system: Objects, attributes, associations
- Sequence diagrams
  - Describe the dynamic behavior between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object
- Activity diagrams
  - Describe the dynamic behavior of a system, in particular the workflow.

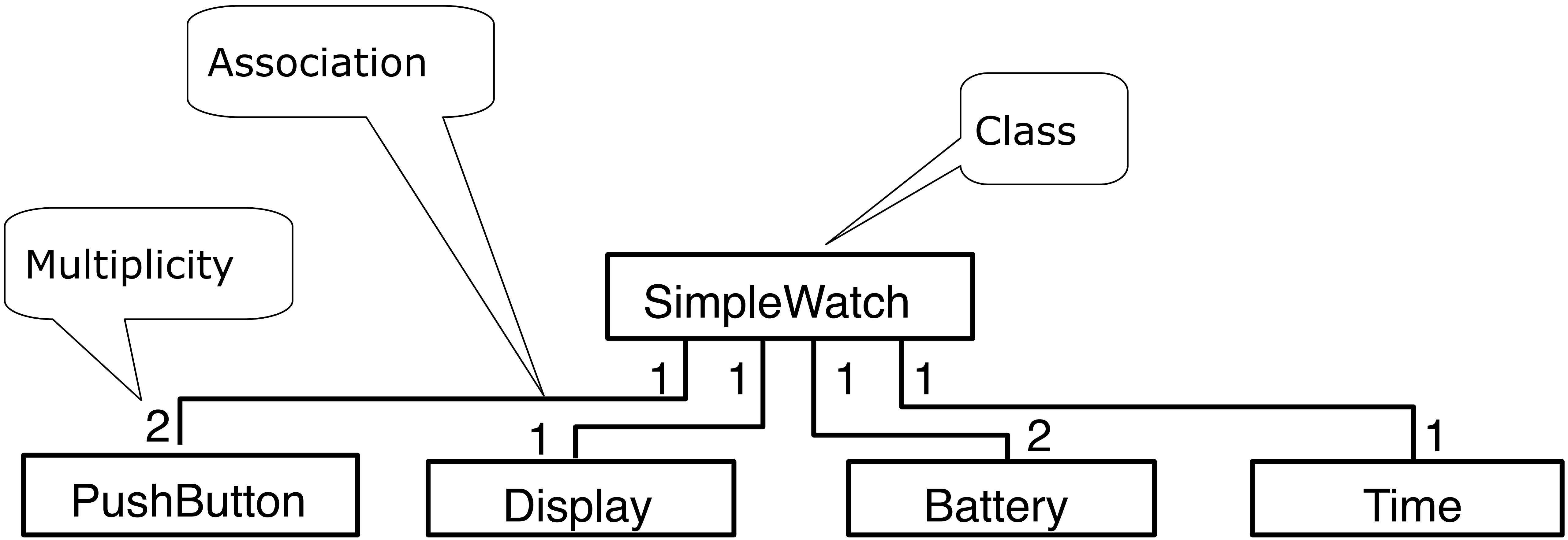
UML Core Conventions

- All UML Diagrams denote graphs of nodes and edges
  - Nodes are entities and drawn as rectangles or ovals
  - Rectangles denote classes or instances
  - Ovals denote functions



- Names of Classes are not underlined
  - SimpleWatch
  - Firefighter
- Names of Instances are underlined
  - myWatch:SimpleWatch
  - Joe:Firefighter
- An edge between two nodes denotes a relationship between the corresponding entities

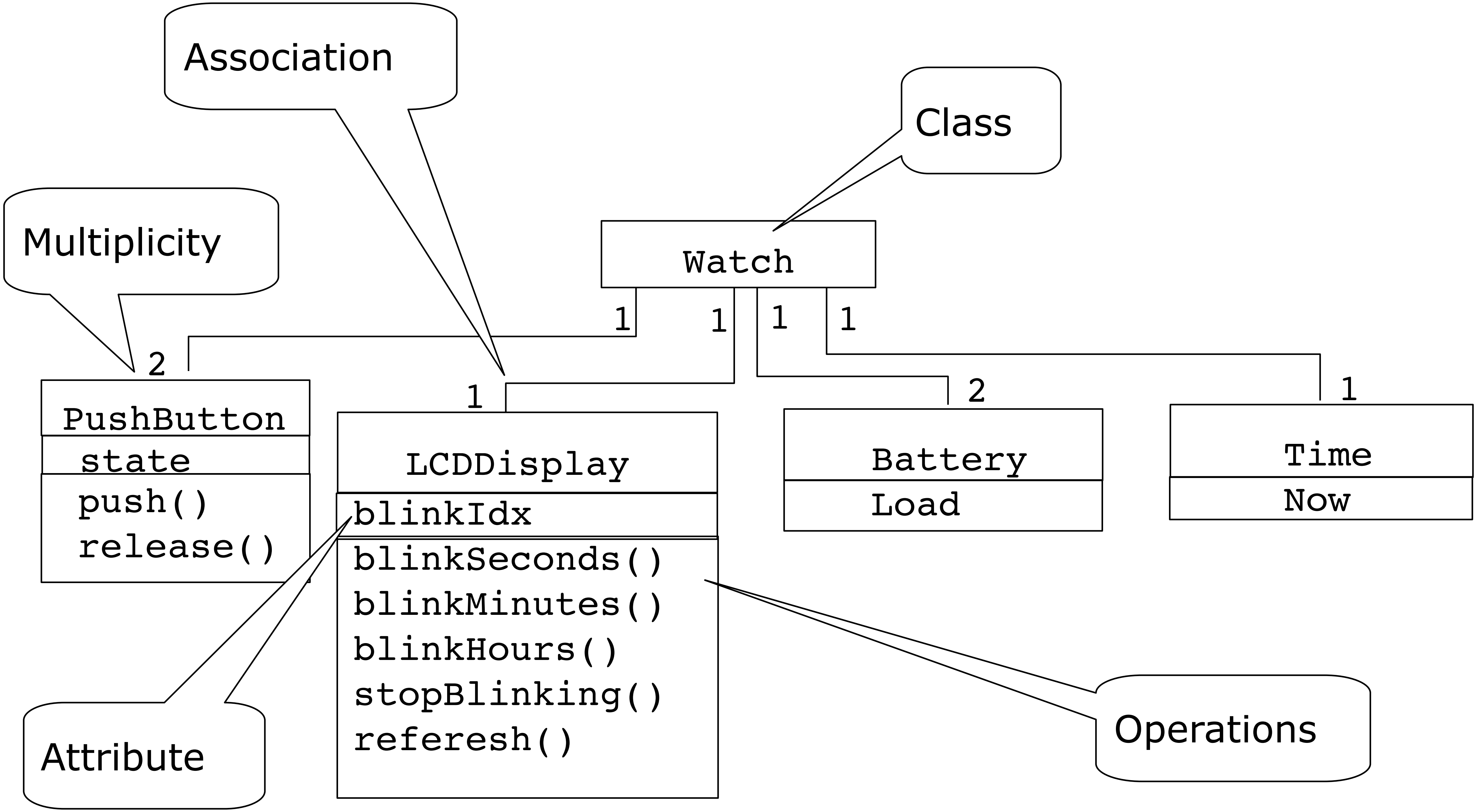
UML first pass: Class diagrams



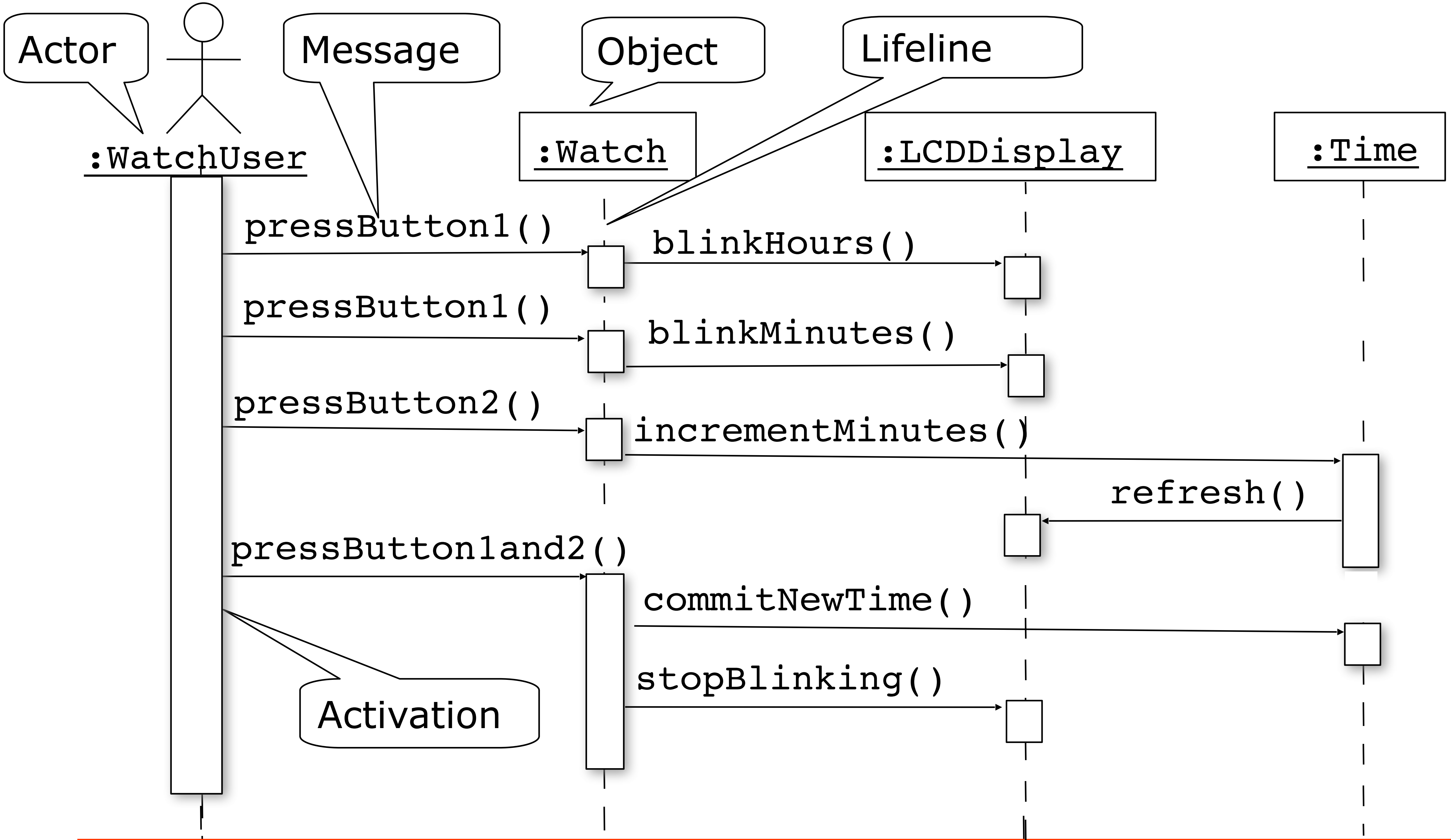
Class diagrams represent the structure of the system

UML first pass: Class diagrams

Class diagrams represent the structure of the system

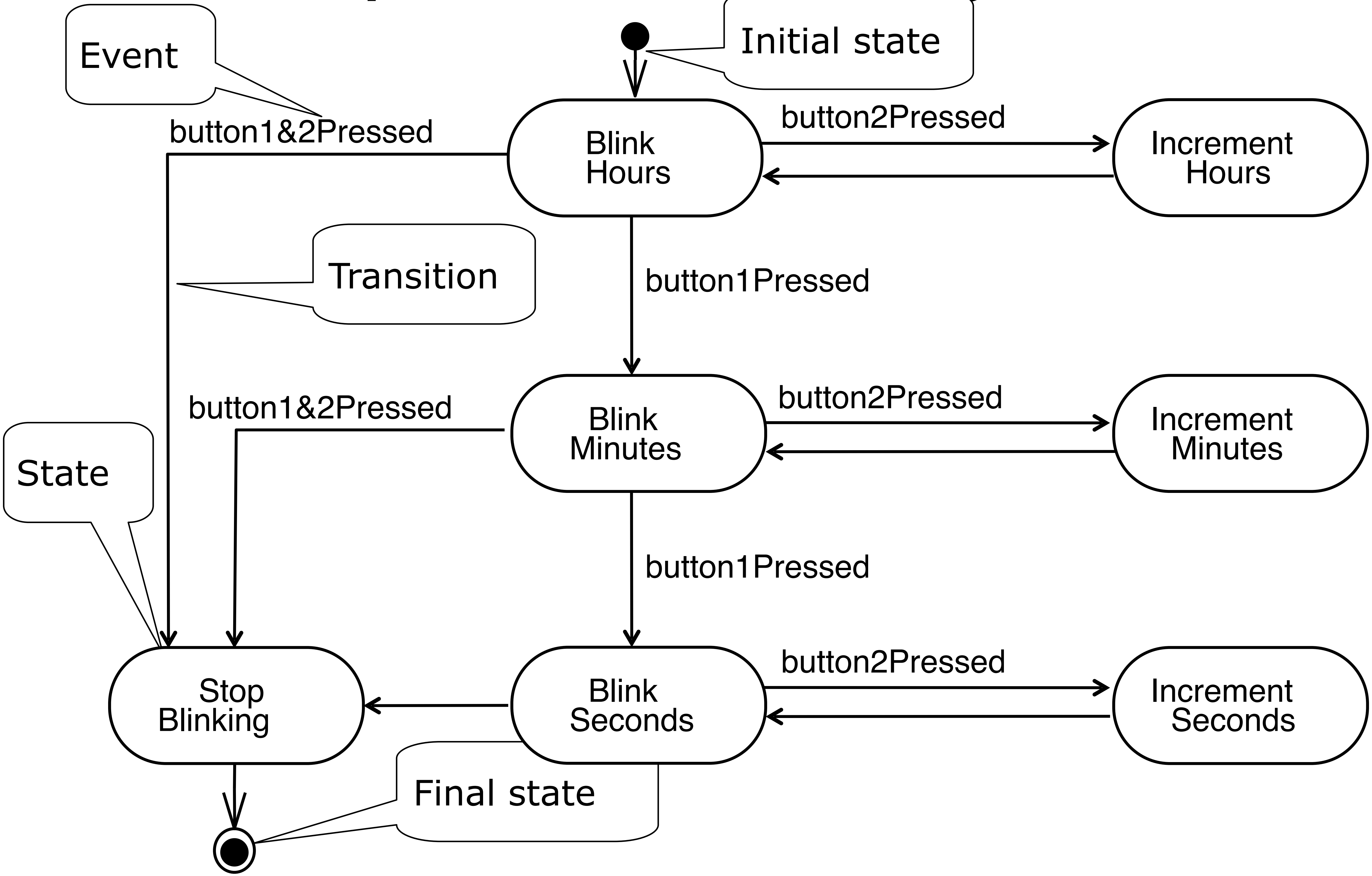


UML first pass: Sequence diagram



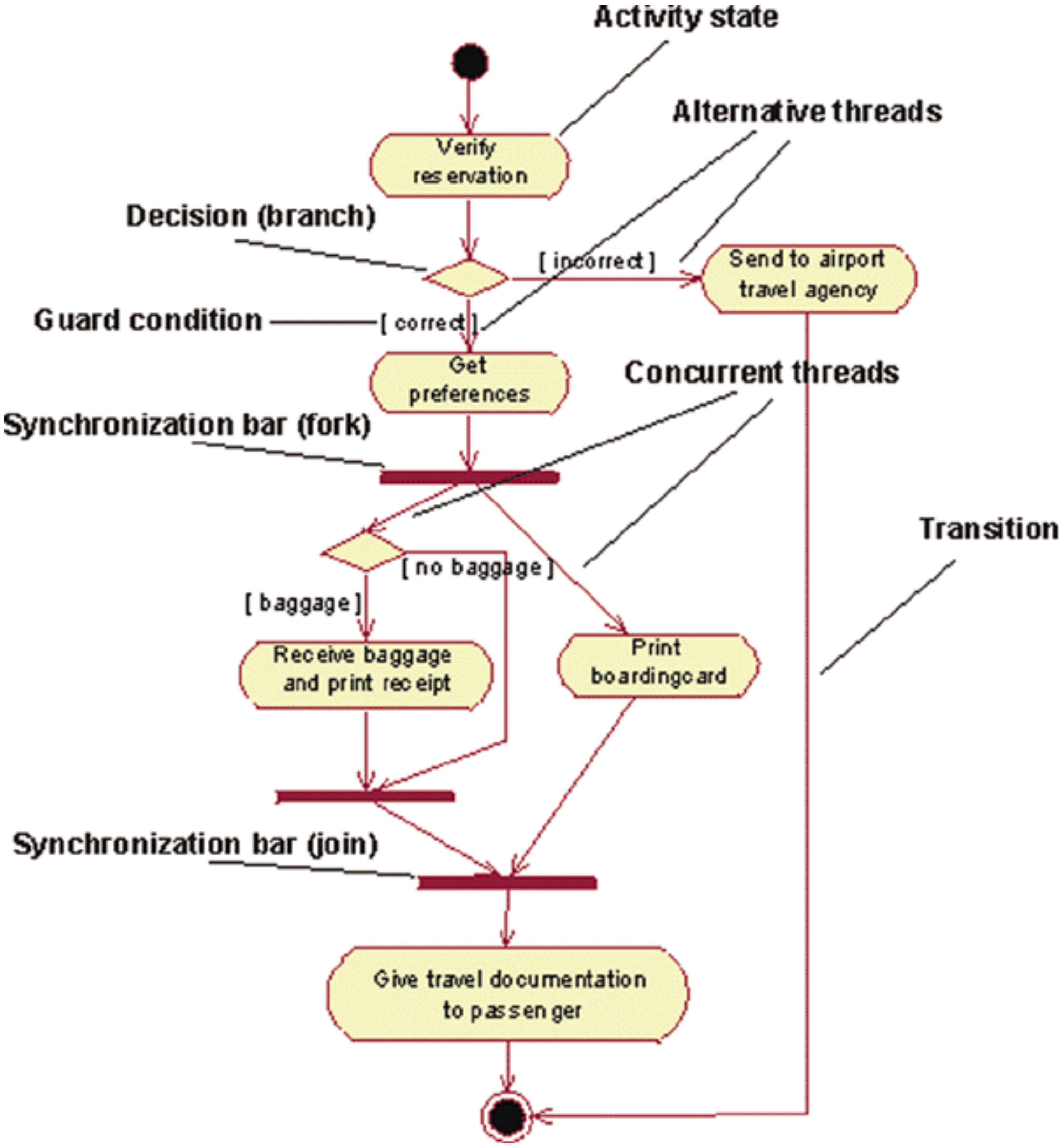
Sequence diagrams represent the behavior of a system as messages ("interactions") between *different objects*

UML first pass: Statechart diagrams



Represent behavior of *a single object* with interesting dynamic behavior.

UML first pass: Activity diagrams



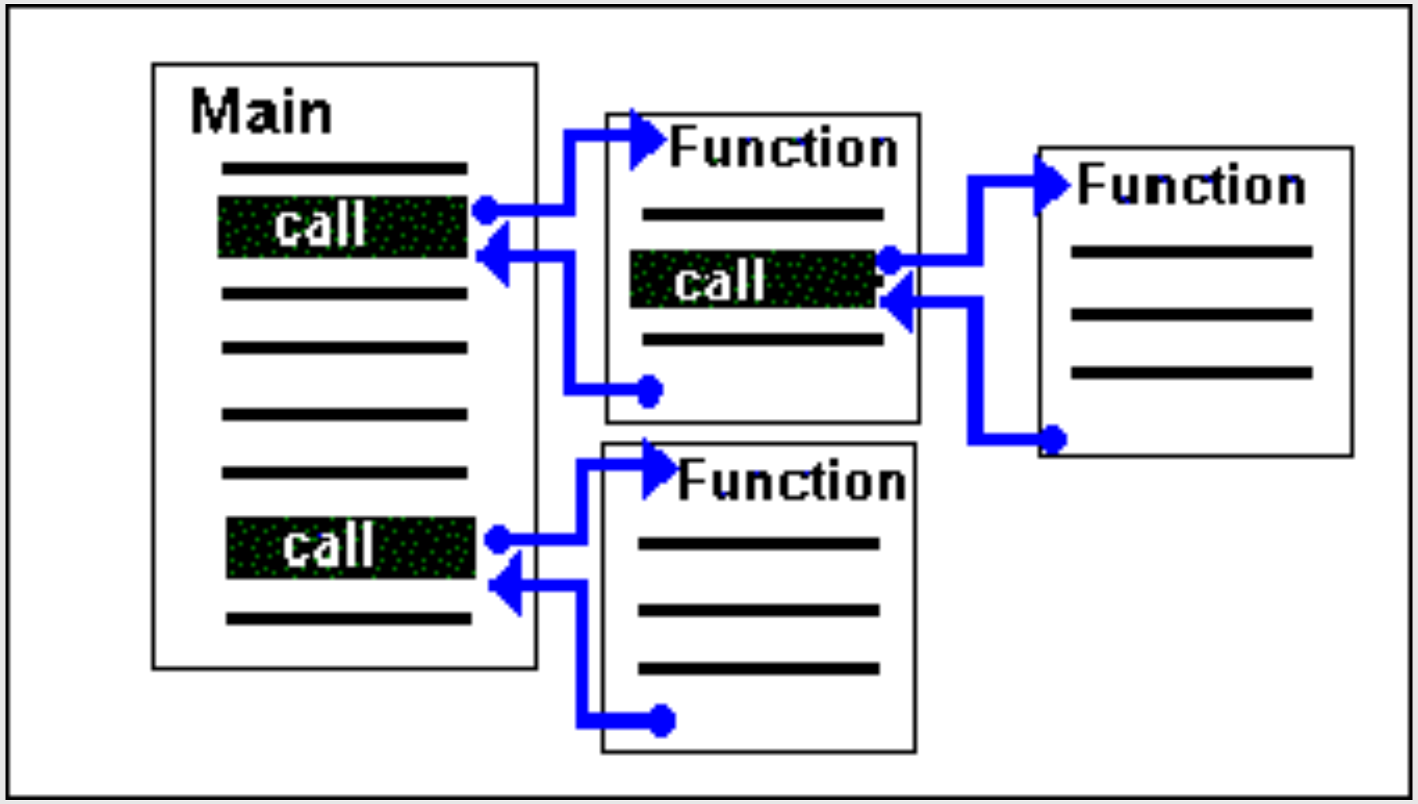
https://www.ibm.com/developerworks/rational/library/2802.html

Represent the dynamic behavior of the system (the workflow).

What is Object Orientation?

Procedural paradigm:

- Software is organized around the notion of *procedures*
- *Procedural abstraction*
  - Works as long as the data is simple



What is Object Orientation?

Object oriented paradigm:

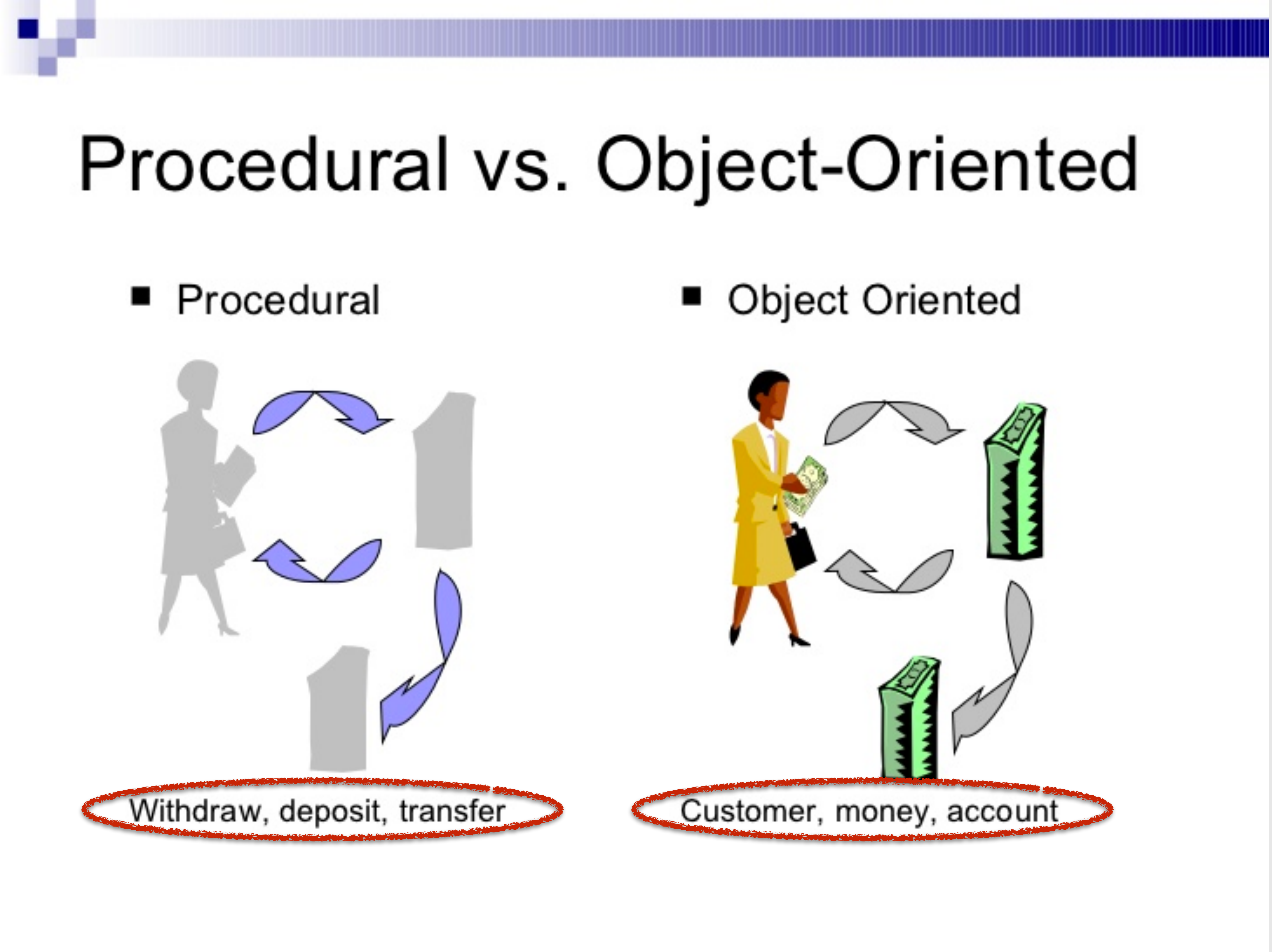
- Organizing procedural abstractions in the context of data abstractions

Object Oriented paradigm

An approach to the solution of problems in which all computations are performed in the context of objects.

- The objects are instances of classes, which:
    - are data abstractions
    - contain procedural abstractions that operate on the objects

- A running program can be seen as a collection of objects collaborating to perform a given task



Procedural vs. Object-Oriented

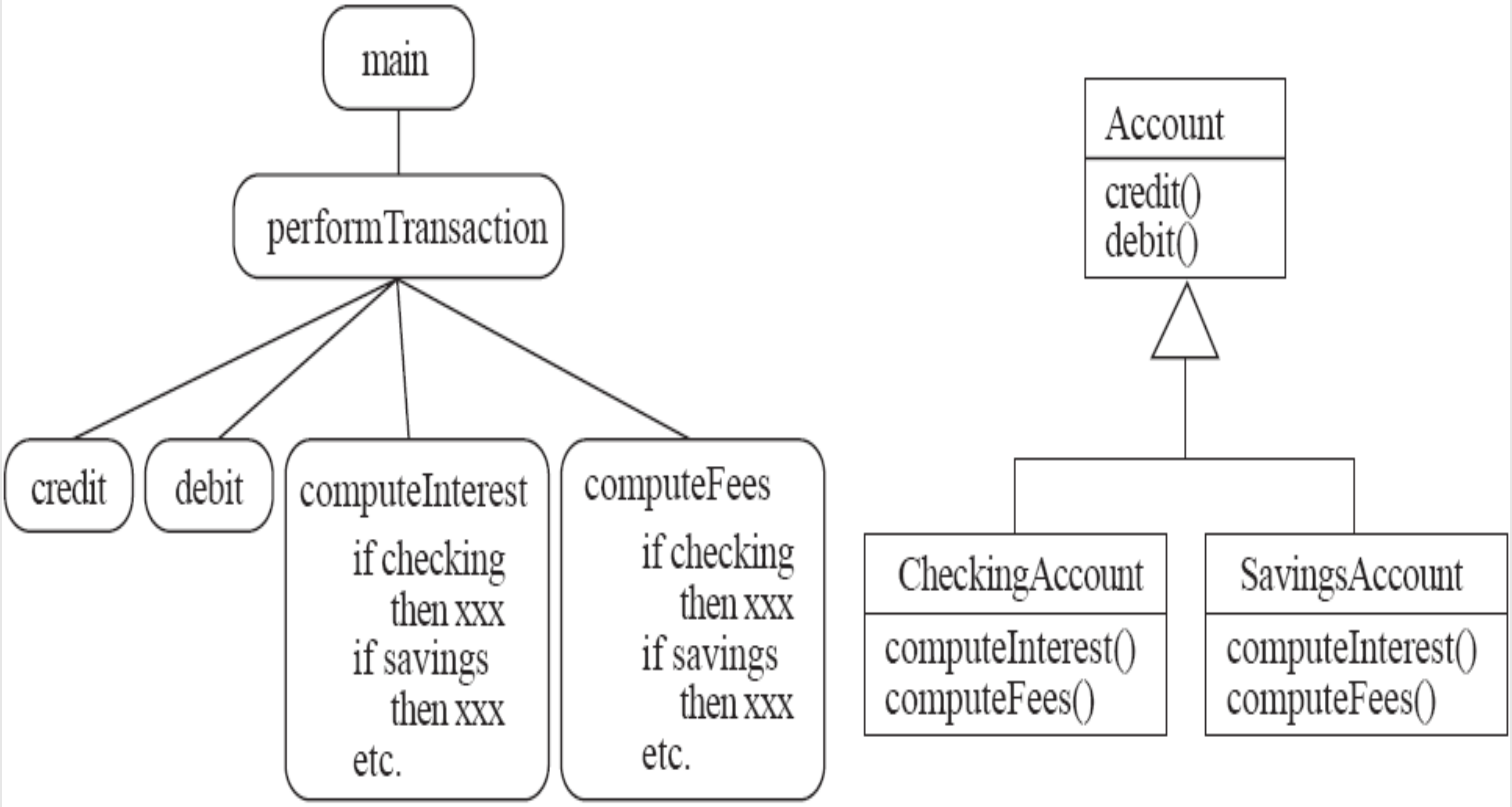
- Procedural
- Object Oriented

Withdraw, deposit, transfer

Customer, money, account

https://www.alphansotech.com/procedural-vs-object-oriented-programming

A View of the Two paradigms

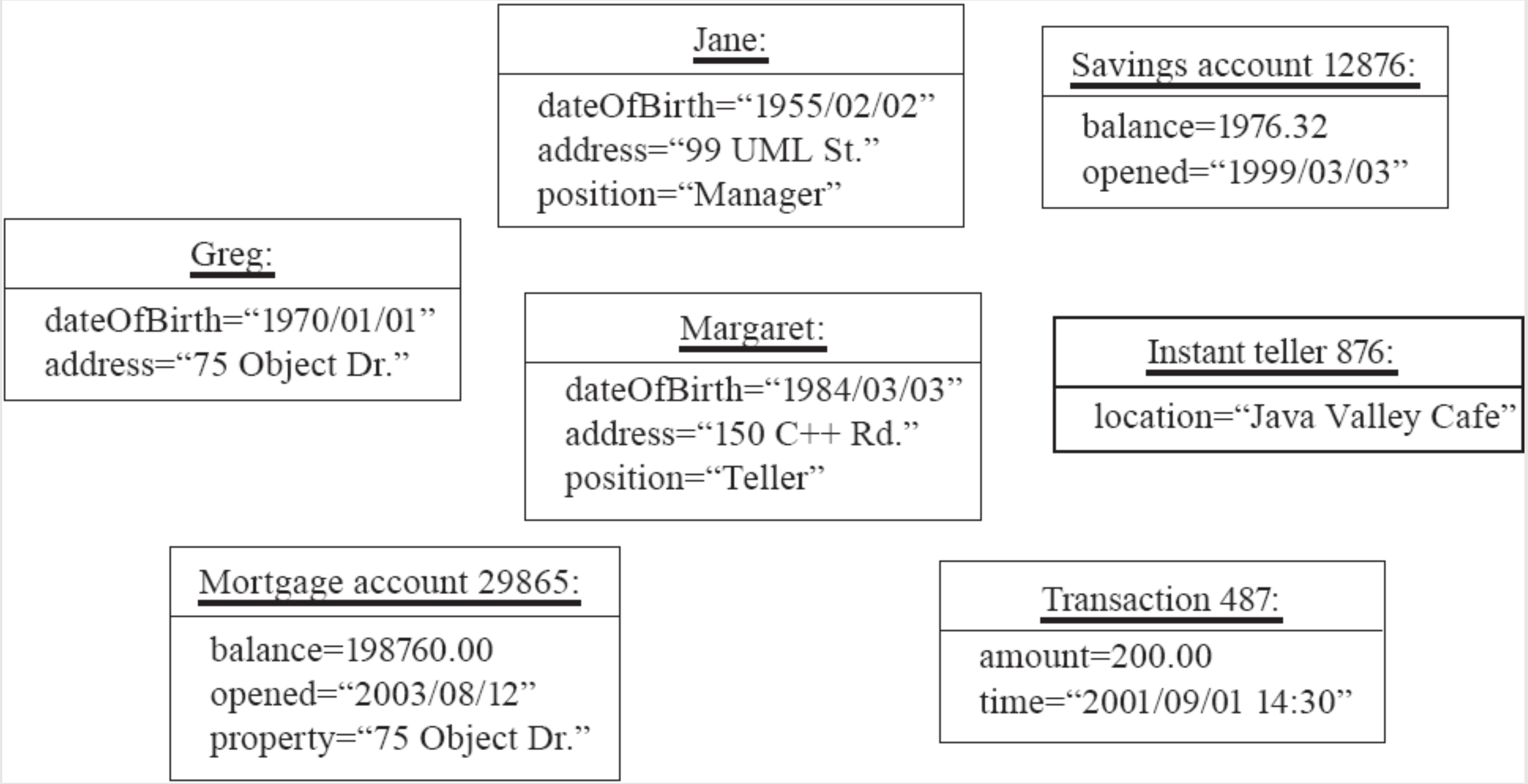


Classes and Objects

Object

- A chunk of structured data in a running software system
- Has *properties*
  - Represent its state
- Has *behaviour*
  - How it acts and reacts
  - May simulate the behaviour of an object in the real world

Objects



Jane:
dateOfBirth="1955/02/02"
address="99 UML St."
position="Manager"

Savings account 12876:
balance=1976.32
opened="1999/03/03"

Greg:
dateOfBirth="1970/01/01"
address="75 Object Dr."

Margaret:
dateOfBirth="1984/03/03"
address="150 C++ Rd."
position="Teller"

Instant teller 876:
location="Java Valley Cafe"

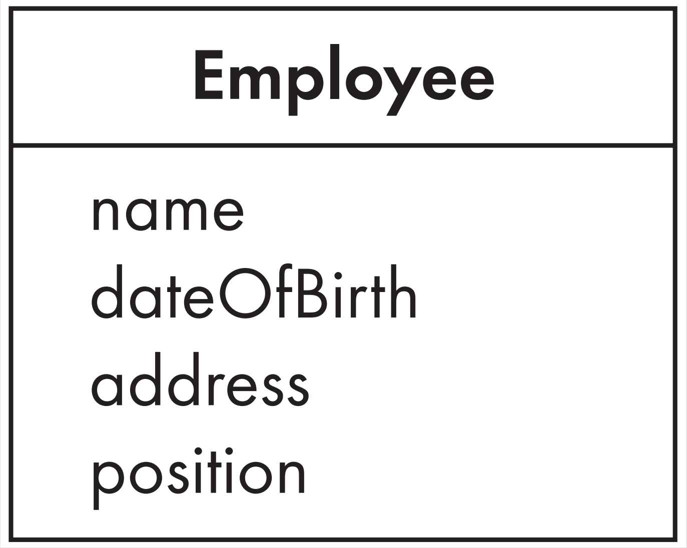
Mortgage account 29865:
balance=198760.00
opened="2003/08/12"
property="75 Object Dr."

Transaction 487:
amount=200.00
time="2001/09/01 14:30"

Classes

A class:

- A unit of abstraction in an object oriented (OO) program
- Represents similar objects
    - Its *instances*
- A kind of software module
    - Describes its instances’ structure (properties)
    - Contains *methods* to implement their behaviour



Is Something a Class or an Instance?

- Something should be a *class* if it could have instances
- Something should be an *instance* if it is clearly a *single* member of the set defined by a class

Film

- Class; instances are individual films.

Reel of Film:

- Class; instances are physical reels

Film reel with serial number SW19876

- Instance of **`ReelOfFilm`**

Science Fiction

- Instance of the class **`Genre`**.

Science Fiction Film

- Class; instances include ‘Star Wars’

Showing of ‘Star Wars’ in the Phoenix Cinema at 7 p.m.:

- Instance of **`ShowingOfFilm`**

Naming classes

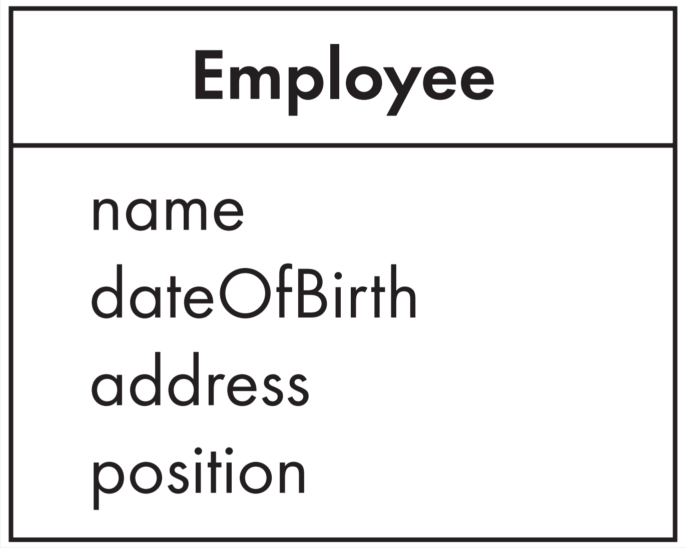
- Use *capital* letters
  - E.g. `BankAccount` not `bankAccount`
- Use *singular* nouns
- Use the right level of generality
  - E.g. `Municipality`, not `City`
- Make sure the name has only *one* meaning
  - E.g. ‘bus’ has several meanings

Variables

Variables defined inside a class corresponding to data present in each instance

- Also called *fields* or *member variables*

- Attributes
  - Simple data
  - E.g. `name`, `dateOfBirth`



Employee

name
dateOfBirth
address
position

Variables vs. Objects

A variable

- *Refers* to an object
- May refer to different objects at different points in time

An object can be referred to by several different variables at the same time

Type of a variable

- Determines what classes of objects it may contain

```
Example: Employee emp1= new Employee();
                  emp1.name= “Omar”;
         Employee  emp2=emp1;
                  emp1= new Employee()
                  emp1.name= “Sarah”;
```

Class variables

A *class variable's* value is *shared* by all instances of a class.

- Also called a *static* variable
- If one instance sets the value of a class variable, then all the other instances see the same changed value.
- Class variables are useful for:
    - Default or 'constant' values (e.g. PI)
    - Lookup tables and similar structures

Caution: *do not over-use class variables*

Organizing Classes into Inheritance Hierarchies

Superclasses

- Contain features common to a set of subclasses

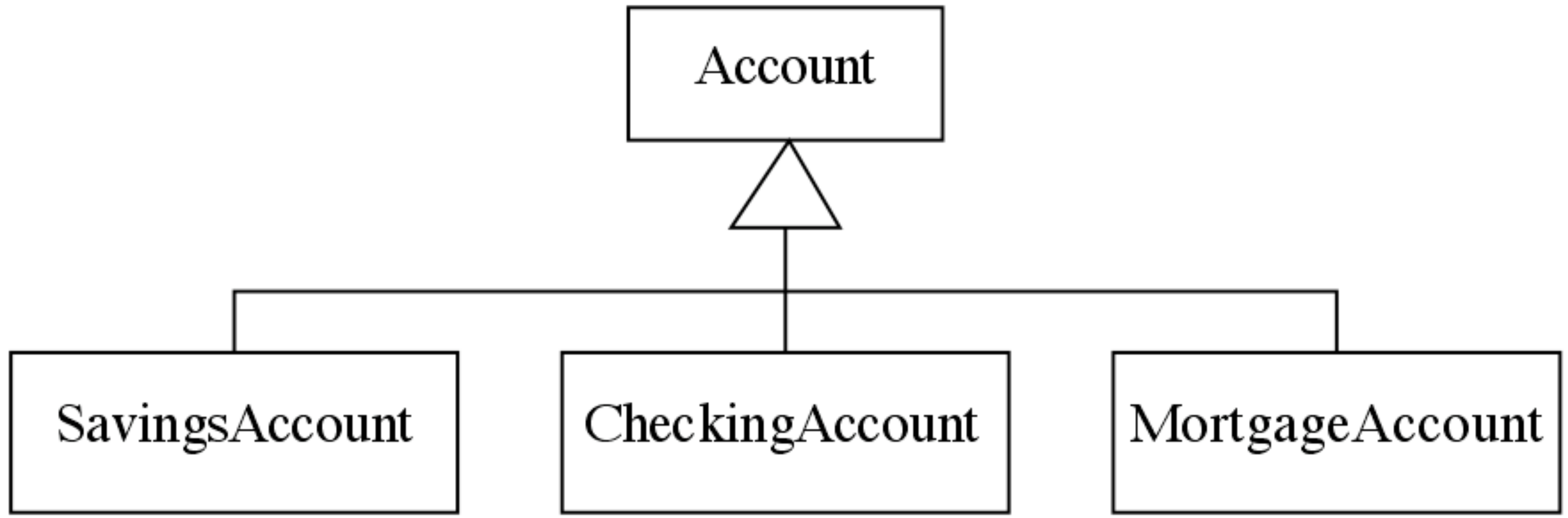
Inheritance hierarchies

- Show the relationships among superclasses and subclasses
- A triangle shows a *generalization*

Inheritance

- The *implicit* possession by all subclasses of features defined in its superclasses

An Example Inheritance Hierarchy

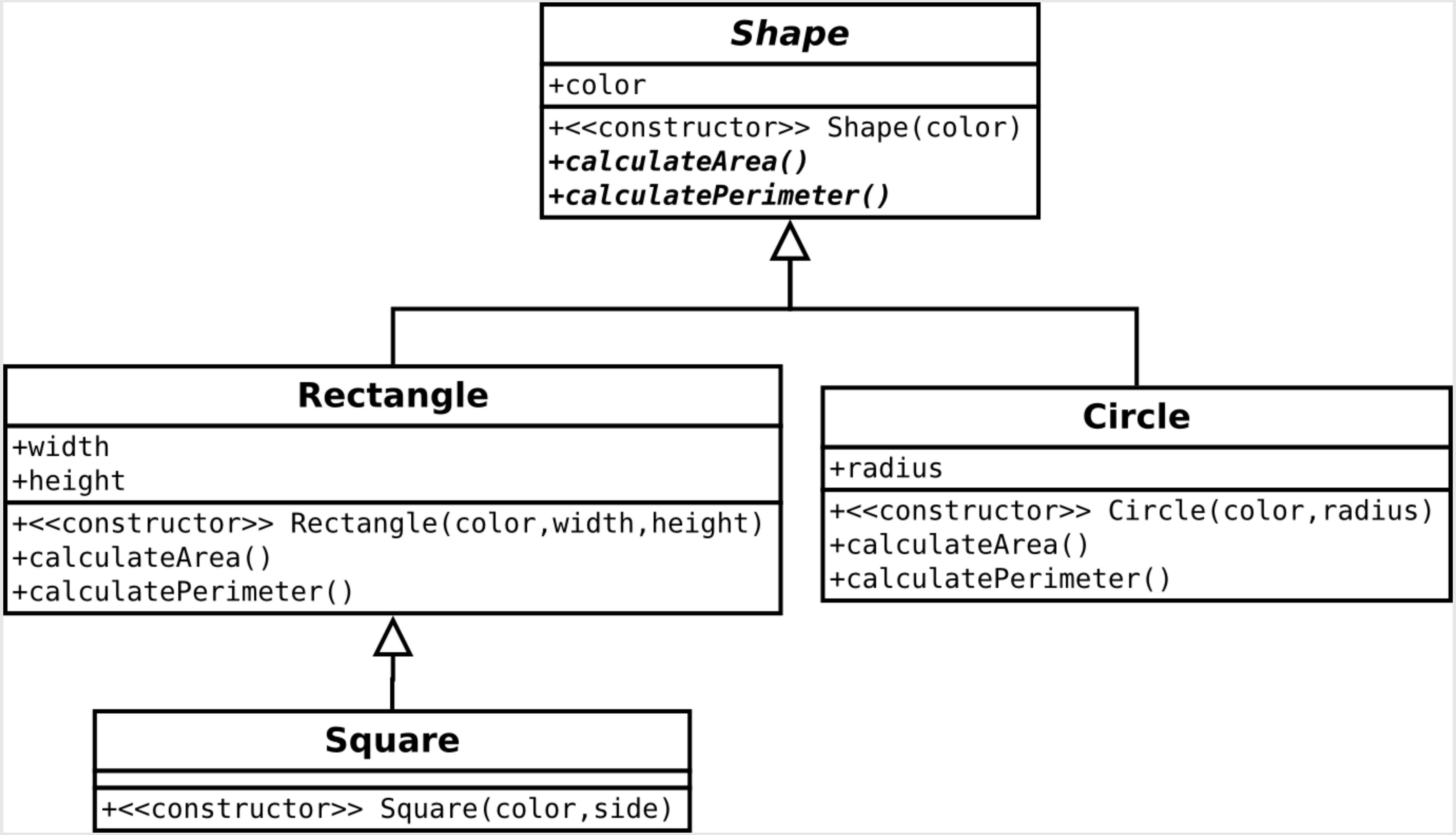


Polymorphism

A property of object oriented software by which an *abstract operation may be performed in different ways* in different classes.

- Requires that there be *multiple methods of the same name*
- The choice of which one to execute depends on the object that is in a variable
- Reduces the need for programmers to code many `if-else` or `switch` statements

Polymorphism



Shape

+color

+<<constructor>> Shape(color)
+calculateArea()
+calculatePerimeter()

Rectangle

+width
+height

+<<constructor>> Rectangle(color,width,height)
+calculateArea()
+calculatePerimeter()

Circle

+radius

+<<constructor>> Circle(color,radius)
+calculateArea()
+calculatePerimeter()

Square

+<<constructor>> Square(color,side)

The Isa Rule

Always check generalizations to ensure they obey the isa rule

- “A checking account ***is an*** account”
- “A village ***is a*** municipality”

Should ‘Province’ be a subclass of ‘Country’?

- No, it violates the isa rule
  - “A province ***is a*** country” is invalid!

Why Java?

⍰Java is a general purpose programming language.

⍰Java is the Internet programming language.

⍰Java is an Object-Oriented Language.

⍰Java is widely used in the market

Java, Web, and Beyond

⍰Java can be used to develop standalone applications.

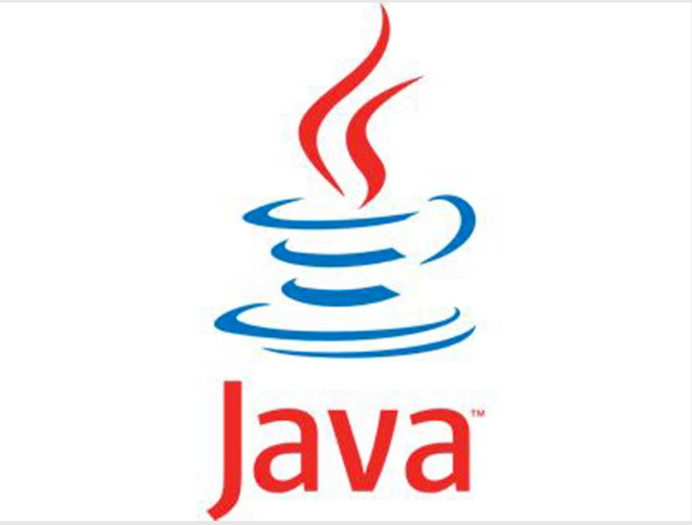
⍰Java can be used to develop applications running from a browser.

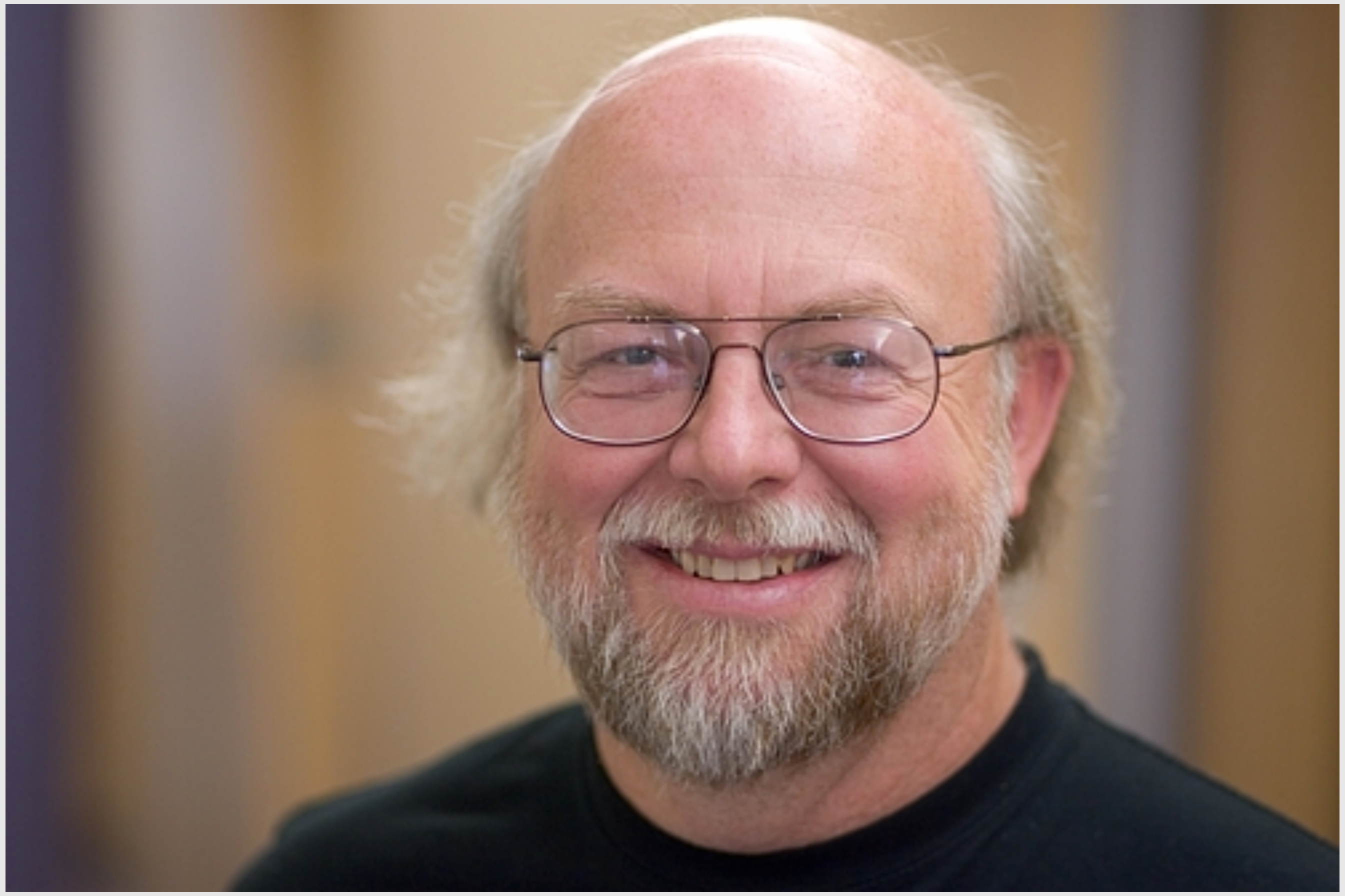
⍰Java can also be used to develop applications for hand-held devices.

⍰Java can be used to develop applications for Web servers.

Java’s History

- James Gosling and Sun Microsystems





http://www.java.com/en/javahistory/index.jsp

Java documentation

Looking up classes and methods is an essential skill

- Looking up unknown classes and methods will get you a long way towards understanding code

Java documentation can be automatically generated by a program called Javadoc

- Documentation is generated from the code and its comments
- You should format your comments as shown in some of the book's examples
    - These may include embeded html

Characters and Strings

`Character` is a class representing Unicode characters

- More than a byte each
- Represent any world language

`char` is a primitive data type containing a Unicode character

`String` is a class containing collections of characters

- `+` is the operator used to concatenate strings

Arrays and Collections

Arrays are of fixed size and lack methods to manipulate them

`ArrayList` is the most widely used class to hold a *collection* of other objects

- More powerful than arrays, but less efficient

`Iterator`s are used to access members of `Vector`s

- Enumerations were formally used, but were more complex

```
a = new ArrayList();
Iterator i = a.iterator();
while(i.hasNext())
{
   aMethod(i.next());
}
```

Casting

Java is very strict about types

- If variable v is declared to have type X, you can only invoke operations on v that are defined in X or its superclasses
    - Even though an instance of a *subclass* of X may be actually stored in the variable
- If you *know* an instance of a subclass is stored, then you can *cast* the variable to the subclass
    - E.g. if I know a `Vector` contains instances of `String`, I can get the next element of its `Iterator` using:

      ```
              (String)i.next();
      ```
    - To avoid casting you could also have used templates:

      ```
      a = ArrayList<String>; i=a.iterator(); i.next()
      ```

Exceptions

Anything that can go wrong should result in the raising of an Exception

- Exception is a class with many subclasses for specific things that can go wrong

Use a try - catch block to trap an exception

```
try
{
   // some code
}
catch (ArithmeticException e)
{
  // code to handle division by zero
}
```

Interfaces

Like abstract classes, but cannot have executable statements

- Define a set of operations that make sense in several classes
- Abstract Data Types

A class can implement any number of interfaces

- It must have concrete methods for the operations

You can declare the type of a variable to be an interface

- This is just like declaring the type to be an abstract class

Important interfaces in Java's library include

- Runnable, Collection, Iterator, Comparable, Cloneable

Packages and importing

A package combines related classes into subsystems

- All the classes in a particular directory

Classes in different packages can have the same name

- Although not recommended

Importing a package is done as follows:

```
import finance.banking.accounts.*;
```

Access control

Applies to methods and variables

- `public`
  - Any class can access
- `protected`
  - Only code in the package, or subclasses can access
- (blank)
  - Only code in the package can access
- `private`
  - Only code written in the class can access
  - Inheritance still occurs!

Programming Style Guidelines

Remember that programs are for people to read

- Always choose the simpler alternative
- Reject clever code that is hard to understand
- Shorter code is not necessarily better

Choose good names

- Make them highly descriptive
- Do not worry about using long names

Programming style …

Comment extensively

- Comment whatever is non-obvious
- Do not comment the obvious
- Comments should be 25-50% of the code

Organize class elements consistently

- Variables, constructors, public methods then private methods

Be consistent regarding layout of code

Programming style …

Avoid duplication of code

- Do not ‘clone’ if possible
  - Create a new method and call it
  - Cloning results in two copies that may both have bugs
    - When one copy of the bug is fixed, the other may be forgotten

Programming style ...

Adhere to good object oriented principles

- E.g. the ‘isa rule’

Prefer `private` as opposed to `public`

Do not mix user interface code with non-user interface code

- Interact with the user in separate classes
  - This makes non-UI classes more reusable

Implicit Import and Explicit Import

```
java.util.* ; // Implicit import

java.util.JOptionPane; // Explicit Import
```

No performance difference