OBJECT-ORIENTED LANGUAGE AND THEORY

2. JAVA BASICS & UML

Nguyen Thi Thu Trang
trangntt@soict.hust.edu.vn

Content

1.1. Identifiers

- Identifiers:
  - A set of characters representing variables, methods, classes and labels
- Naming rules:
  - Characters can be numbers, alphabets, '$' or '_'
  - Name must not:
    - Start by a Number
    - Be the same as a keyword
  - Distinguish between UpperCases and LowerCases
    - Yourname, yourname, YourName and yourName are for different identifiers

An_Identifier
a_2nd_Identifier
Go2
$10

An-Identifier
2nd_Identifier
goto
10$

1.1. Identifiers (2)

- Naming convention:
  - Start with an Alphabet
  - Package: all in lowercase
    - Theexample
  - Class: the first letter of word is in uppercase
    - TheExample
  - Method/field: start with a lowercase letter, the first letter of remaining word is in uppercase
    - theExample
  - Constants: All in uppercase
    - THE_EXAMPLE

5

1.1. Identifiers (3)

- Literals
  `null true false`
- Keyword

```
abstract assert boolean break byte case catch
char class continue default do double else
extends final finally float for if implements
import instanceof int interface long native new
package private protected public return short
static strictfp super switch synchronized this
throw throws transient try void volatile while
```

- Reserved for future use

```
byvalue cast const future generic goto inner operator
outer rest var volatile
```

6

1.2. Data Types

- Two categories:
  - Primitive
    - Integer
    - Float
    - Char
    - Logic (boolean)
  - Reference
    - Array
    - Object

7

a. Primitives

- Every variable must be declared with a data type:
  - Primitive data type contains a single value
  - Size and format must be appropirate to its data type
- Java has 4 primitive data types

Categories:

a. **integer**

b. **floating point**

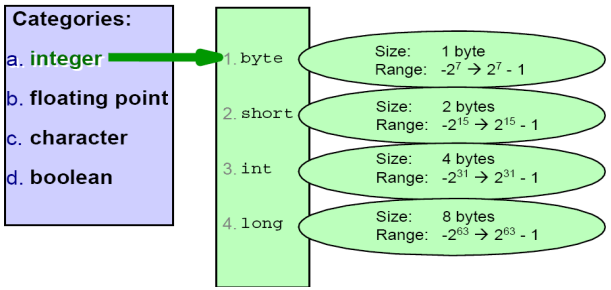
c. **character**

d. **boolean**

8

Integer

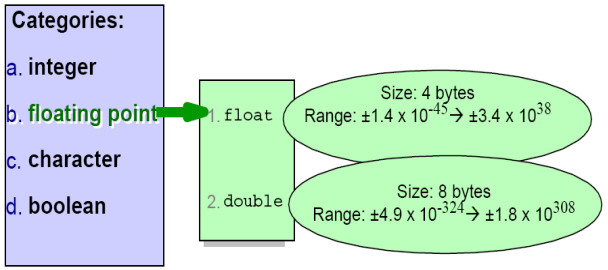
- Signed integer
- Initially created with value 0



9

Real

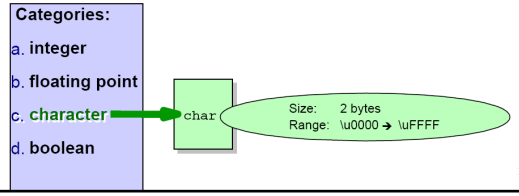
- Initially created with value 0.0



10

Characters

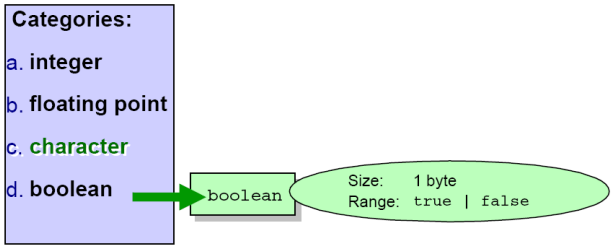
- Unsigned Unicode characters, placed between two single quotes
- 2 ways to assign value:
  - Using number in hexa: char uni ='\u05D0';
  - Using character: char a = 'A' ;
- Default value is zero (\u0000)



11

Logic value

- Boolean value is clearly specified in Java
  - An int value can not be used for a boolean value
  - Can store value "true" or "false"
- Boolean variable is initially created with false value



12

b. Literal

- Literal is a value of a set of primitive data types and character string.
- Has 5 categories:
  - integer
  - floating point
  - boolean
  - character
  - string

Literals
integer……..7
floating point…7.0f
boolean…….true
character…….'A'
string……."A"

13

Literal of Integer

- Octals start with number 0
  - 032 = 011 010(2) = 16 + 8 + 2 = 26(10)
- Hexadecimals start with number 0 and character x
  - 0x1A = 0001 1010(2) = 16 + 8 + 2 = 26(10)
- Ends with character "L" reperesenting data type long
  - 26L
- Uppercase characters, usually have the same values
  - 0x1a , 0x1A , 0X1a , 0X1A đều có giá trị 26 trong hệ decimal

14

Literal of Real

- **`Float`** ends with character **`f`** (or **`F`**)
  - `7.1f`
- **`Double`** ends with character **`d`** (or **`D`**)
  - `7.1D`
- **`e`** (or **`E`**) is used in scientific representation:
  - `7.1e2`
- A value without ending character is considered as a **`double`**
  - `7.1` giống như `7.1d`

15

Literal of boolean, character and string

- boolean:
  - `true`
  - `false`
- Character:
  - Located between two single quotes
  - Example: 'a', 'A' or '\uffff'
- String:
  - Located between two double quotes
  - Example: "Hello world", "Xin chao ban",...

16

Escape sequence

- Characters for keyboard control
  - **`\b backspace`**
  - **`\f form feed`**
  - **`\n newline`**
  - **`\r return`** (về đầu dòng)
  - **`\t tab`**
- Display special characters in a string
  - **`\" quotation mark`**
  - **`\' apostrophe`**
  - **`\\ backslash`**

17

c. Casting

- Java is a strong-type languge
  - Casting a wrong type to a variable can lead to a compiler error or exceptions in JVM
- JVM can implicitly cast a data type to a larger data type
- To cast a variable to a narrower data type, we need to do it explicitly

```
int a, b;
short c;
a = b + c;
```

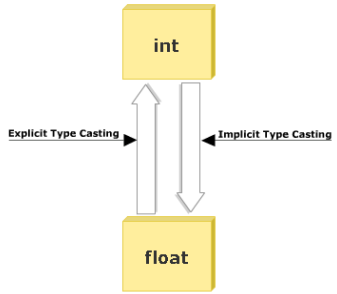
```
int d;
short e;
e = (short)d;
```

```
double f;
long g;
f = g;
g = f; //error
```

18

c. Casting (2)

- Casting is done automatically if no information loss occurs
  - byte → short → int → long → float → double
- Eexplicit cast is required if there is a "risk" of reduced accuracy



19

Example - Casting

```
long p = (long) 12345.56; // p == 12345
int g = p;    // syntax error although int
              // can hold a value of 12345
char c = 't';
int j = c;    // implicit casting
short k = c; // error
short k = (short) c; // explicit casting
float f = 12.35; // error
```

20

d. Declaration and Creation of Variables

- Simple variables (that are not array) need to be initialized before being used in expressions
  - Can declare and initialize at the same time.
  - Using = to assign (including initialization)
    - Example:

```
int i, j;    // Variable declaration
i = 0;
int k =i+1;
float x=1.0f, y=2.0f;
System.out.println(i);  // Print out 0
System.out.println(k);  // Print out 1
System.out.println(j);  // Compile error
```

21

Comments

- Java supports three types of comments:
  - // Comments in a single line
    // Without line break
  - /* Comments as a paragraph */
  - /** Javadoc * comments in form of Javadoc */

22

Command

- Command ends with;
- Multiple commands can be written on one line
- A command can be written in multiple lines
  - Example:

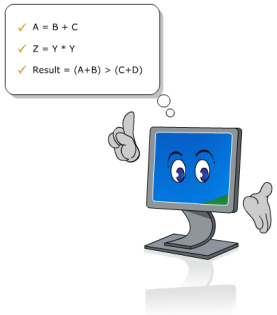
```
System.out.println(
            "This is part of the same line");
```

```
a=0; b=1; c=2;
```

23

1.3. Operators

- Combining single values or child expressions into a new expression, more complex and can return a value.
- Java provides the following operators:
  - Arithmetic operators
  - Bit oprator, Relational opretors
  - Logic operators
  - Assignment operators
  - Unary operators



24

1.3. Operators (2)

- Arithmetic operators
  - +, -, *, /, %
- Bit operators
  - AND: &, OR: |, XOR: ^, NOT: ~
  - bit: <<, >>
- Relational operators
  - ==, !=, >, <, >=, <=
- Logic operators
  - &&, ||, !

25

1.3. Operators (3)

- Unary operators
  - Reverse sign : +, -
  - Increase/decrease by 1 unit: ++, --
  - Negation of a logic expression: !
- Assignment operators
  - =, +=, -=, %= similar to >>, <<, &, |, ^

26

Priorities of Operators

- Define the order of performing operators – are identified by parentheses or by default as follows:
  - Postfix operators `[] .` (params) `x++ x--`
  - Unary operators `++x --x +x -x ~ !`
  - Creation or cast `new (type)x`
  - Multiplicative `* / %`
  - Additive `+ -`
  - Shift `<< >> >>> (unsigned shift)`
  - Relational `< > <= >= instanceof`
  - Equality `== !=`
  - Bitwise AND `&`
  - Bitwise exclusive OR `^`
  - Bitwise inclusive OR `|`
  - Logical AND `&&`
  - Logical OR `||`
  - Conditional (ternary) `?:`
  - Assignment `= *= /= %= += -= >>= <<= >>>= &= ^= |=`

27

1.4. if – else statement

- Syntax

```
if (condition){
   statements;
}
else {
   statements;
}
```

- condition expression can receive boolean value
- else expression is optional

28

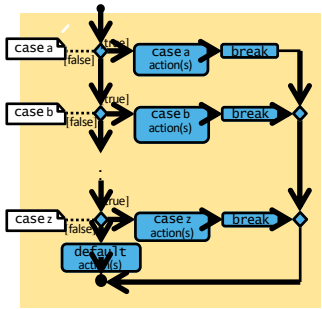
Example – Checking odd – even numbers

```
class CheckNumber
{
 public static void main(String args[])
 {
   int num =10;
   if (num %2 == 0)
          System.out.println (num+ "la so chan");
   else
          System.out.println (num + "la so le");
 }
}
```

29

1.5. switch – case statement

- Checking a single variable with different values and perform the corresponding case
  - break: exits switch-case command
  - Default: manages values outside the values defined in case:



30

Example - switch - case

```
switch (day) {
  case 0:
  case 1:
      rule = "weekend";
      break;
  case 2:
      …
  case 6:
      rule = "weekday";
      break;
  default:
      rule = "error";
}
```

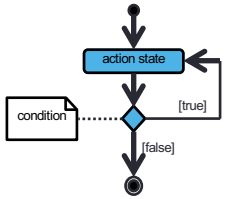
```
if (day == 0 || day == 1) {
      rule = "weekend";
} else if (day > 1 && day <7) {
      rule = "weekday";
} else {
      rule = error;
}
```

31

1.6. while and do while statements

- Perform a command or a command block while the control expression is still true
  - while() performs 0 or multiple times
  - do...while() performs at least 1 time

```
int x = 2;
while (x < 2) {
        x++;
        System.out.println(x);
}
```




```
int x = 2;
do {
        x++;
        System.out.println(x);
} while (x < 2);
```

32

Example – while loop

```
class WhileDemo{
 public static void main(String args[]){
   int a = 5,fact = 1;
   while  (a >= 1){
        fact *=a;
        a--;
    }
   System.out.println("The Factorial of 5
                          is "+fact);
 }
}
```

33

1.7. for loop

- Syntax:

```
for (start_expr; test_expr; increment_expr){
 // code to execute repeatedly
}
```

  - 3 expressions can be absent
  - A variable can be declared in for command:
    - Usually declare a counter variable
    - Usually declare in "start" expression
    - Variable scope is in the loop
- Example:

```
for (int index = 0; index < 10; index++) {
 System.out.println(index);
}
```

34

Example - for loop

```
class ForDemo
{
 public static void main(String args[])
 {
     int i=1, sum=0;
     for (i=1;i<=10;i+=2)
          sum+=i;
     System.out.println ("Sum of first five
                    old numbers is " + sum);
 }
}
```

35

Commands for changing control structure

- break
  - Can be used to exit switch command
  - Terminate loops for, while or do...while
  - There are two types:
    - Labeling: continue to perform commands after the labeled loop
    - No-Labeling: perform next commands outside the loop
- continue
  - Can be used for for, while or do...while loops
  - Ignore the remaining commands of the current loop and perform the next iteration.

36

Example - break and continue

```
public int myMethod(int x) {
 int sum = 0;
 outer: for (int i=0; i<x; i++) {
     inner: for (int j=i; j<x; j++){
          sum++;
          if (j==1) continue;
          if (j==2) continue outer;
          if (i==3) break;
          if (j==4) break outer;
     }
 }
 return sum;
}
```

37

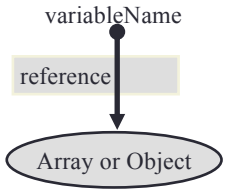
Variable scope

- Scope of a variable is a program area in which that variable is referred to
  - Variables declared in a method can only be accessed inside that method
  - Variables declared in a loop or code block can only be accessed in that loop or that block

```
int a = 1;
for (int b = 0; b < 3; b++){
    int c = 1;
    for (int d = 0; d <3; d++){
        if (c < 3) c++;
    }                                  abcd
    System.out.print(c);
    System.out.println(b);             abc
}
a = c; // ERROR! c is out of scope     a
```

38

1.5. Array



- Finite set of elements of the same type
- Must be declared before use
- Declaration:
  - Syntax:
    - `datatype[] arrayName= new datatype[ARRAY_SIZE];`
    - `datatype arrayName[] = new datatype[ARRAY_SIZE];`
  - Example:
    - `char c[] = new char[12];`

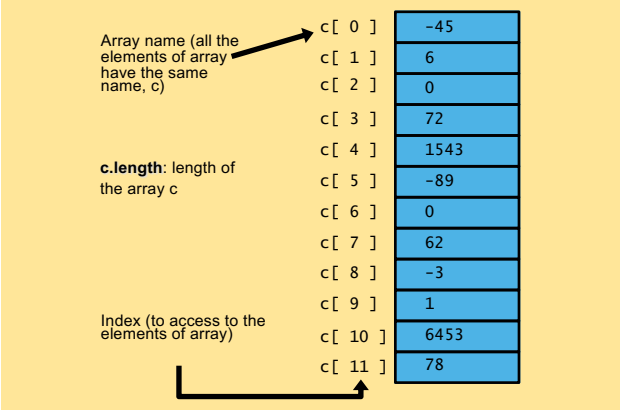
39

Array declaration and initialization

- Declaration and initialization:
  - Syntax:
    - `datatype[] arrayName = {initial_values};`
  - Example:
    - `int[] numbers = {10, 9, 8, 7, 6};`
- Without initialization → receives the default value depending on the data type.
- Always starts with the element of index 0

40

Example - Array



41

Array declaration and initialization – Example

```
int MAX = 5;
boolean bit[] = new boolean[MAX];
float[] value = new float[2*3];
int[] number = {10, 9, 8, 7, 6};
System.out.println(bit[0]); // prints "false"
System.out.println(value[3]); // prints "0.0"
System.out.println(number[1]); // prints "9"
```

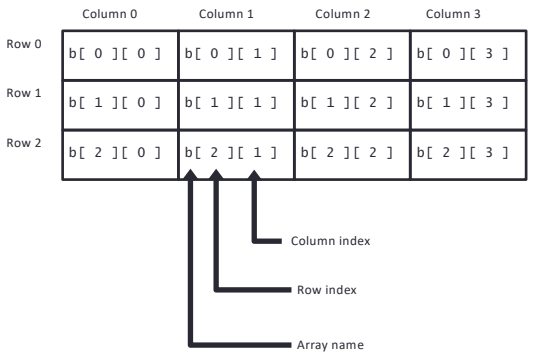
42

Multi-dimensional array

- Table with rows and columns
  - Usually use two-dimensional array
  - Example of declaration b[2][2]
  - `int b[][] = { { 1, 2 }, { 3, 4 } };`
    - 1 and 2 are initialized for b[0][0] and b[0][1]
    - 3 and 4 are initialized for b[1][0] and b[1][1]
  - `int b[3][4];`

43

Multi-dimensional array (2)



@NGUYỄN Thị Thu Trang, trangntt@soict.hust.edu.vn 44

Content

1. Java basics
2. Unified Modeling Language (UML)

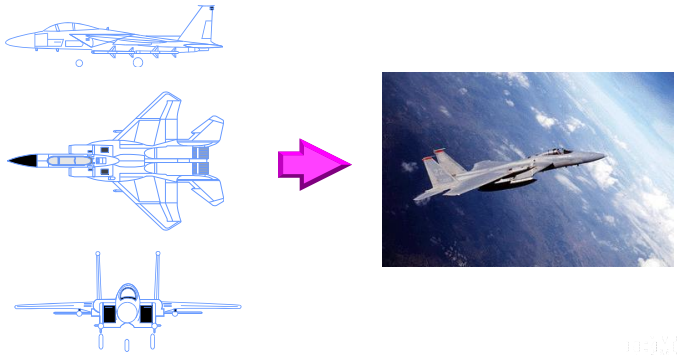
Discussion

- You have a complicated object in the real world, e.g. an airplane

- How can you make it?
- How can you know its structure / design?
- …

2.1. What Is a Model?

- A model is a simplification of reality.

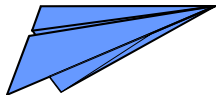


Why Model?

- Modeling achieves four aims:
  - Helps you to **visualize** a system as you want it to be.
  - Permits you to specify the **structure** or **behavior** of a system.
  - Gives you a **template** that guides you in constructing a system.
  - **Documents** the **decisions** you have made.
- You build models of complex systems because you cannot comprehend such a system in its entirety.
- You build models to better understand the system you are developing.

Discussion

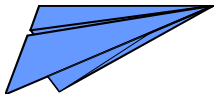
- How do you build a paper airplane?
- If it cannot fly, what will you do?



- What about a fighter jet?

The Importance of Modeling

Less Important ⟷ More Important



Paper Airplane

Fighter Jet

Software Teams Often Do Not Model

- Many software teams build applications approaching the problem like they were building paper airplanes
  - Start coding from project requirements
  - Work longer hours and create more code
  - Lacks any planned architecture
  - Doomed to failure
- Modeling is a common thread to successful projects

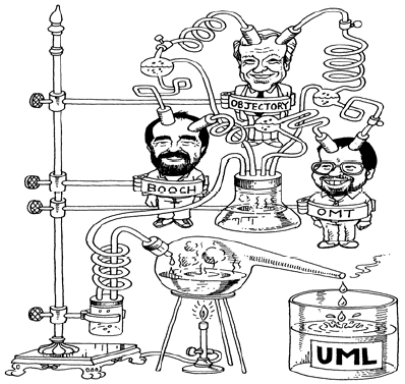
2.2. Why UML?

- 1980s: classical structural analysis and design
- 1990s: object-oriented analysis and design
- Mid-1990s: > 50 object-oriented methods with many design formats (*similar meta-models*)
  - Fusion, Shlaer-Mellor, ROOM, Class-Relation,Wirfs-Brock, Coad-Yourdon, MOSES, Syntropy, BOOM, OOSD, OSA, BON, Catalysis, COMMA, HOOD, Ooram, DOORS…

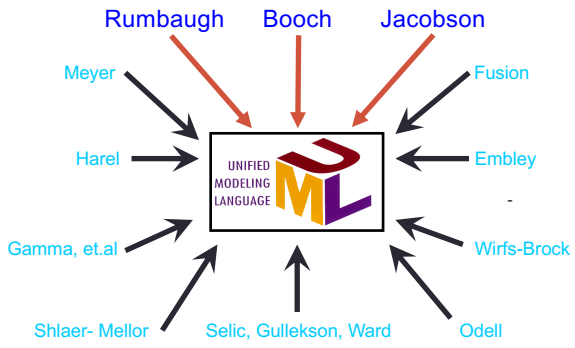
➔ A **unified** modeling language is **indispensable**

UML is a standardization to a single unified language

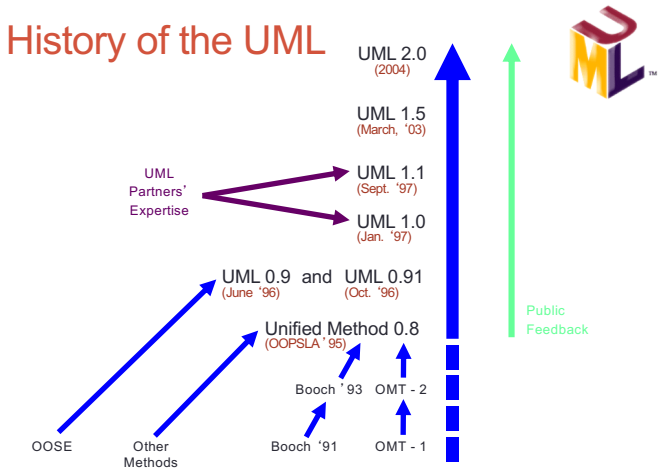
- An Object Management Group (OMG) standard.
- By 3 experts in Rational Software
  - Booch91 (Grady Booch): Conception, Architecture
  - OOSE (Ivar Jacobson): Use cases
  - OMT (Jim Rumbaugh): Analysis



Inputs to the UML



History of the UML



2.3. What Is the UML?

- The UML is a language for
  - Visualizing
  - Specifying
  - Constructing
  - Documenting

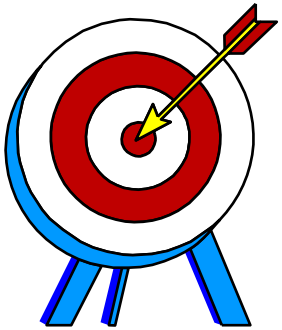
  the artifacts of a software-intensive system.

The UML Is a Language for Visualizing

- Communicating conceptual models to others is prone to error unless everyone involved speaks the same language.
- There are things about a software system you can't understand unless you build models.
- An explicit model facilitates communication.

The UML Is a Language for Specifying

- The UML builds models that are precise, unambiguous, and complete.

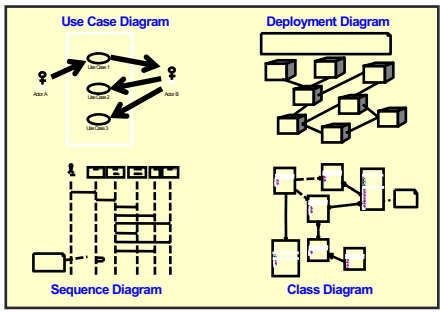


The UML Is a Language for Constructing

- UML models can be directly connected to a variety of programming languages.
  - Maps to Java, C++, Visual Basic, and so on
  - Tables in a RDBMS or persistent store in an OODBMS
  - Permits forward engineering
  - Permits reverse engineering

The UML Is a Language for Documenting

- The UML addresses documentation of system architecture, requirements, tests, project planning, and release management.

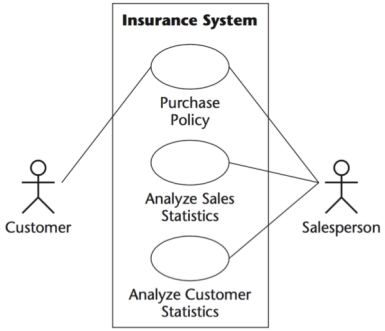


2.4. Common diagrams in UML

- Use-case diagram
- Class diagram
- Object Diagram
- State machine
- Activity diagram
- Interaction diagrams
- Deployment diagram

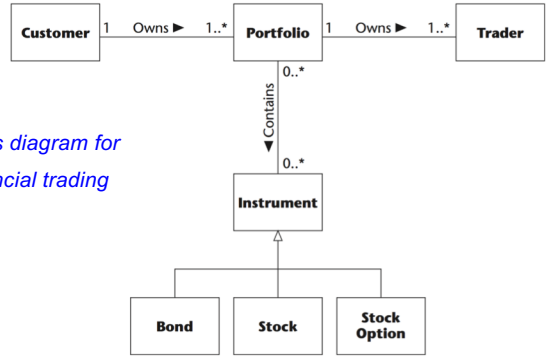
Use case diagram

- A number of external actors and their connection to the use cases that the system provides



Class diagram

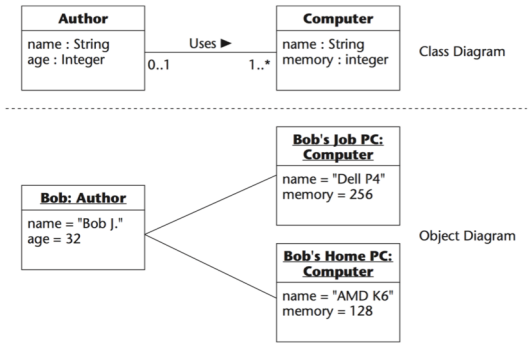
- Static structure of classes in the system



Class diagram for Financial trading

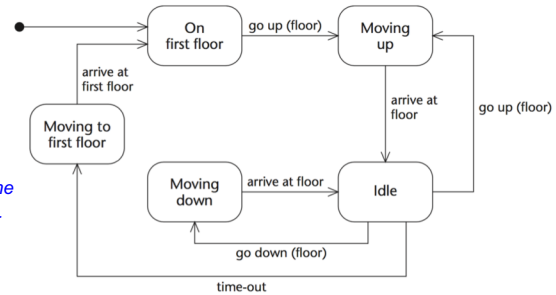
Object diagram

- Shows a number of object instances of classes, instead of the actual classes



State machine

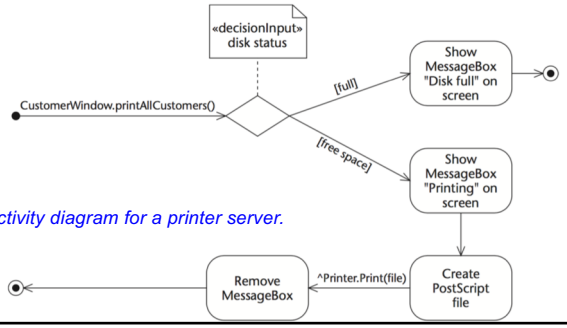
- Shows all the possible states that objects of the class can have during a life-cycle instance, and which events cause the state to change



A state machine for an elevator

Activity diagram

- Shows a sequential flow of actions to describe
  - the activities performed in a general process workflow
  - or other activity flows, such as a use case or a detailed control flow

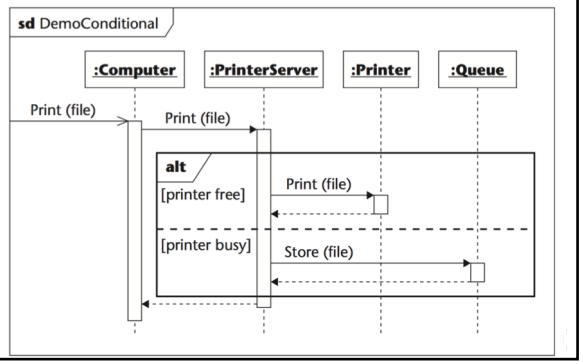


An activity diagram for a printer server.

Interaction Diagrams

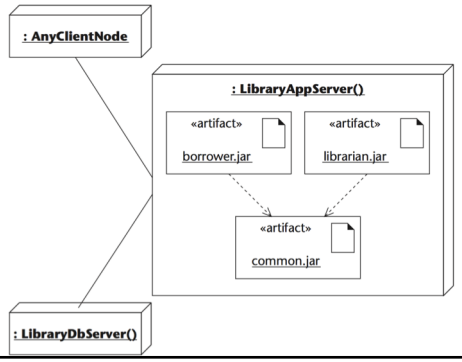
- Show the interaction between objects during the execution of the software

A sequence diagram for a print server



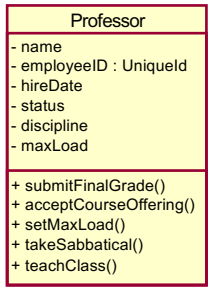
Deployment Diagram

- Shows the physical architecture of the hardware and software in the system



Classes in the UML

- A class is represented using a rectangle with three compartments:
  - The class name
  - The structure (attributes)
  - The behavior (operations)



Classes and Objects

- A class is an abstract definition of an object
  - It defines the structure and behavior of each object in the class.
  - It serves as a template for creating objects.
- Classes are not collections of objects