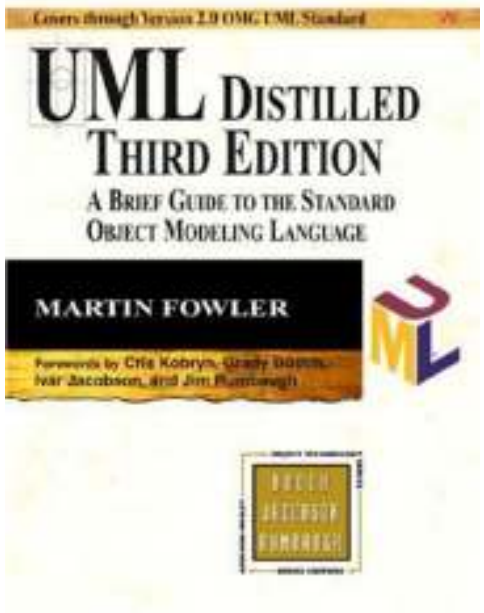


Design and UML Class Diagrams

Suggested reading:

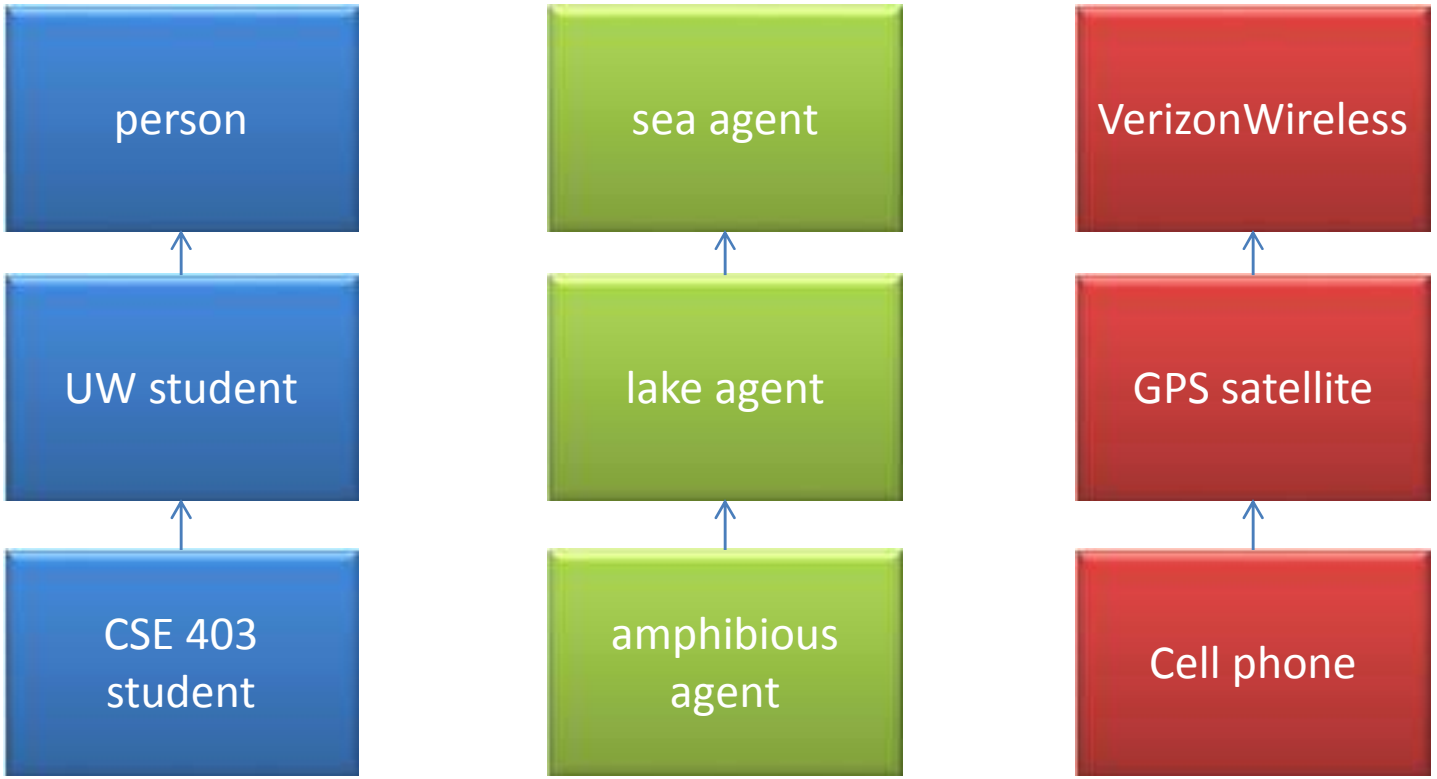
Practical UML: A hands on introduction for developers

http://dn.codegear.com/article/31863

UML Distilled Ch. 3, by M. Fowler

How do people draw / write down software architectures?

Example architectures



Big questions

- What is UML?
  - Why should I bother? Do people really use UML?
- What is a UML class diagram?
  - What kind of information goes into it?
  - How do I create it?
  - When should I create it?

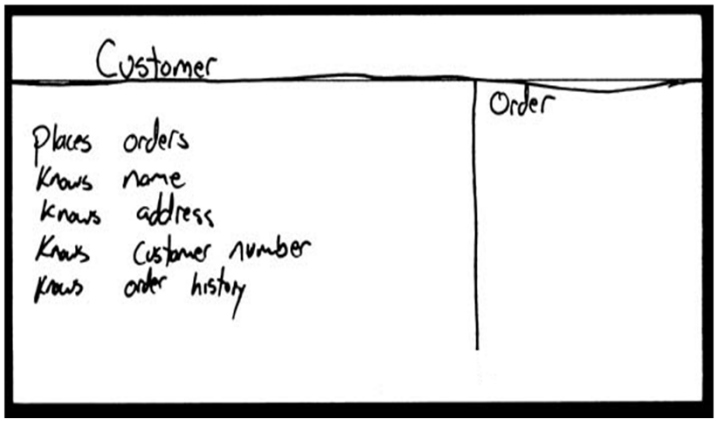
Design phase

- **design**: specifying the structure of how a software system will be written and function, without actually writing the complete implementation

- a transition from "what" the system must do, to "how" the system will do it
  - What classes will we need to implement a system that meets our requirements?
  - What fields and methods will each class have?
  - How will the classes interact with each other?

How do we design classes?

- class identification from project spec / requirements
  - nouns are potential classes, objects, fields
  - verbs are potential methods or responsibilities of a class
- CRC card exercises
  - write down classes' names on index cards
  - next to each class, list the following:
    - **responsibilities**: problems to be solved; short verb phrases
    - **collaborators**: other classes that are sent messages by this class (asymmetric)
- UML diagrams
  - class diagrams (today)
  - sequence diagrams
  - ...



What is UML?

- UML: pictures of an OO system
  - programming languages are not abstract enough for OO design
  - UML is an open standard; lots of companies use it

- What is legal UML?
  - a *descriptive* language: rigid formal syntax (like programming)
  - a *prescriptive* language: shaped by usage and convention
  - it's okay to omit things from UML diagrams if they aren't needed by team/supervisor/instructor

Uses for UML

- as a sketch: to communicate aspects of system
  - forward design: doing UML before coding
  - backward design: doing UML after coding as documentation
  - often done on whiteboard or paper
  - used to get rough selective ideas

- as a blueprint: a complete design to be implemented
  - sometimes done with CASE (Computer-Aided Software Engineering) tools

- as a programming language: with the right tools, code can be auto-generated and executed from UML
  - only good if this is faster than coding in a "real" language

UML

In an effort to promote Object Oriented designs, three leading object oriented programming researchers joined ranks to combine their languages:

- Grady Booch (BOOCH)
- Jim Rumbaugh (OML: object modeling technique)
- Ivar Jacobsen (OOSE: object oriented software eng)

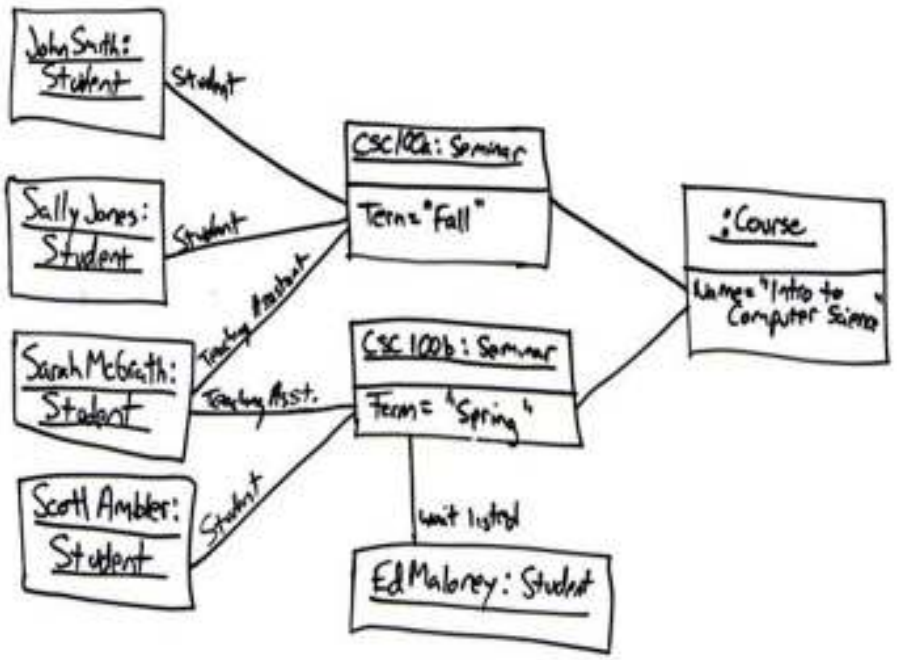
and come up with an industry standard [mid 1990's].

UML – Unified Modeling Language

- Union of all Modeling Languages
  - Use case diagrams
  - Class diagrams
  - Object diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Statechart diagrams
  - Activity diagrams
  - Component diagrams
  - Deployment diagrams
  - ….
- Very big, but a nice standard that has been embraced by the industry.

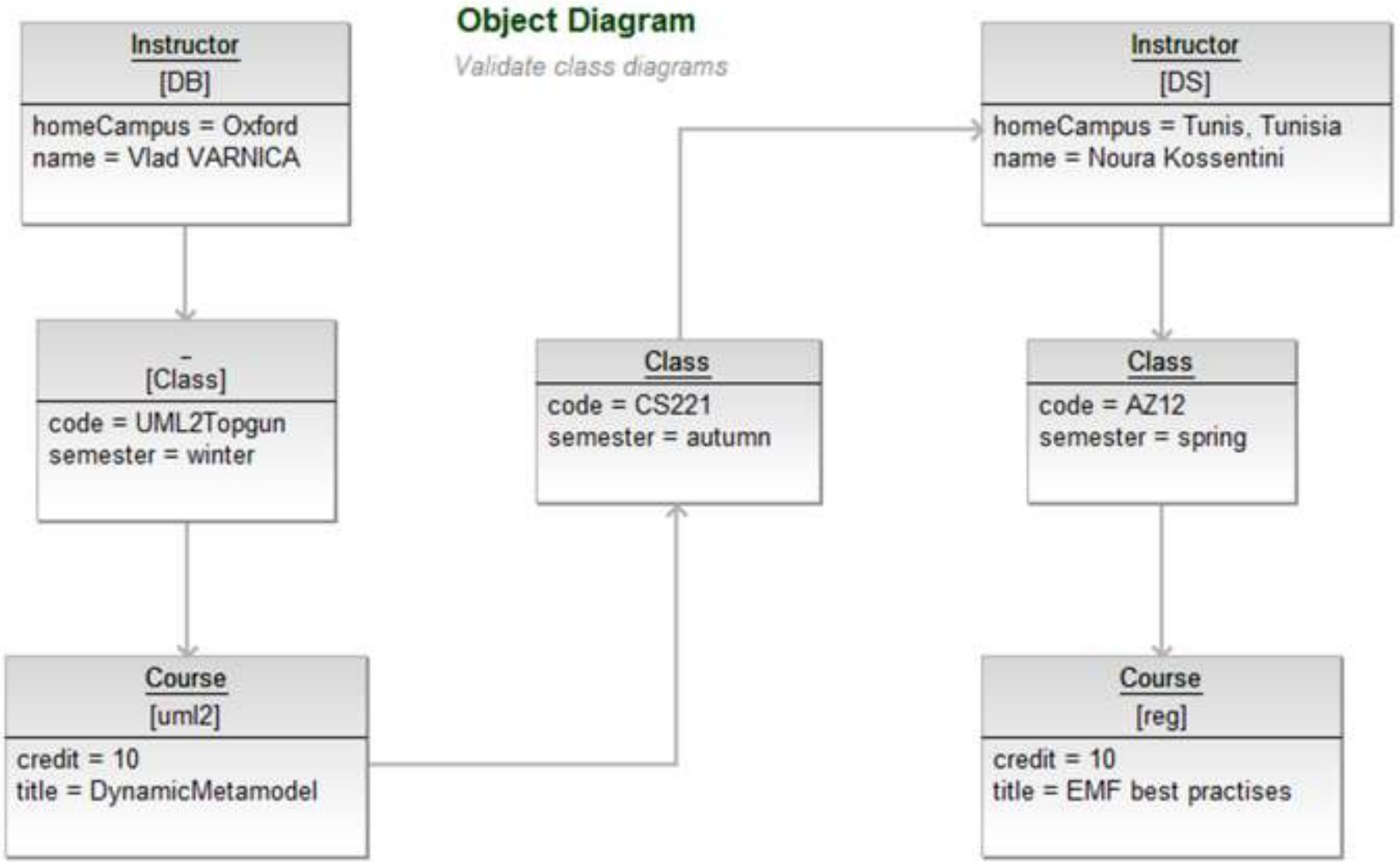
Object diagram (≠ class diagram)

- individual objects (heap layout)
  - objectName : type
  - attribute = value
- lines show field references



- Class diagram:
  - summary of all possible object diagrams

Object diagram example

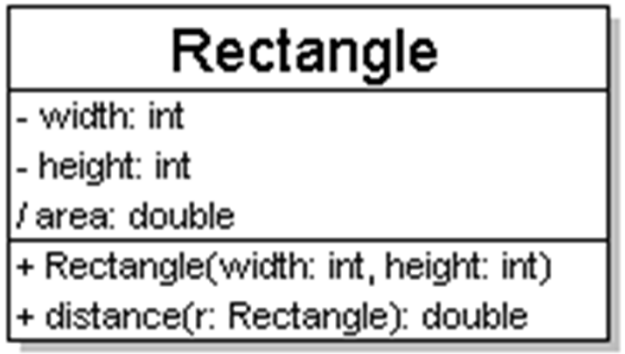


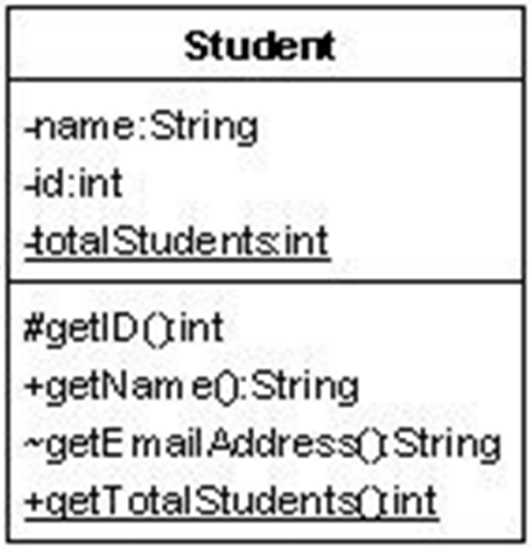
UML class diagrams

- **UML class diagram**: a picture of
  - the classes in an OO system
  - their fields and methods
  - connections between the classes
    - that interact or inherit from each other
- Not represented in a UML class diagram:
  - details of how the classes interact with each other
  - algorithmic details; how a particular behavior is implemented

Diagram of one class

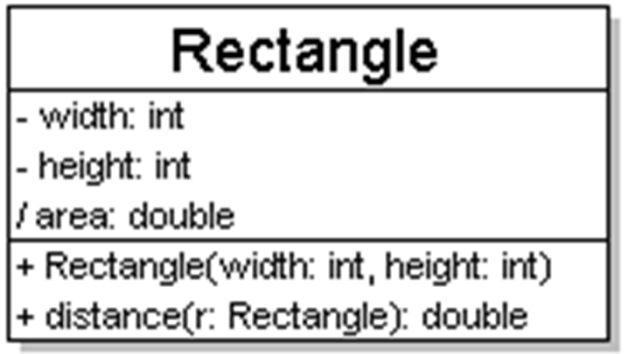
- class name in top of box
  - write <<interface>> on top of interfaces' names
  - use *italics* for an *abstract class* name
- attributes (optional)
  - should include all fields of the object
- operations / methods (optional)
  - may omit trivial (get/set) methods
    - but don't omit any methods from an interface!
  - should not include inherited methods

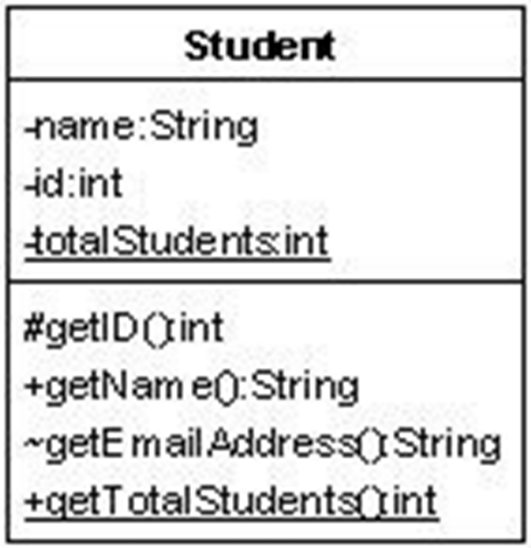




Class attributes (= fields)

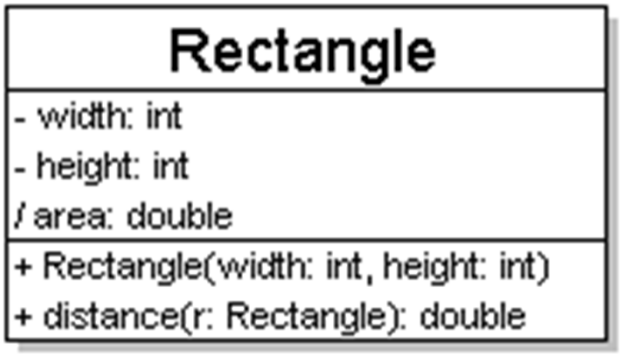
- attributes (fields, instance variables)
  - *visibility name* : *type* [*count*] = *default_value*
  - visibility:
    - \+ public
    - # protected
    - \- private
    - ~ package (default)
    - / derived
  - underline static attributes
  - **derived attribute**: not stored, but can be computed from other attribute values
    - "specification fields " from CSE 331
  - attribute example:
    - balance : double = 0.00

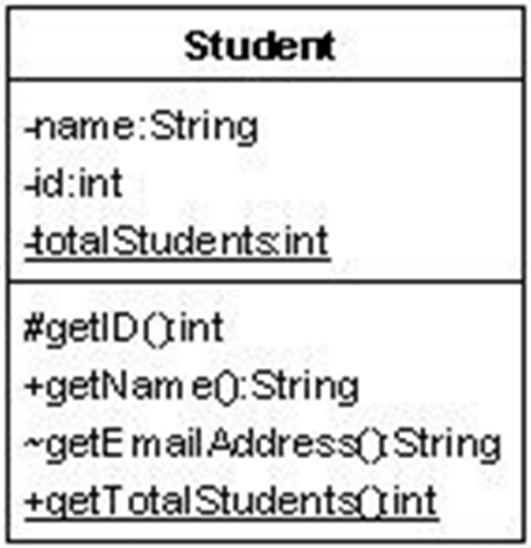




Class operations / methods

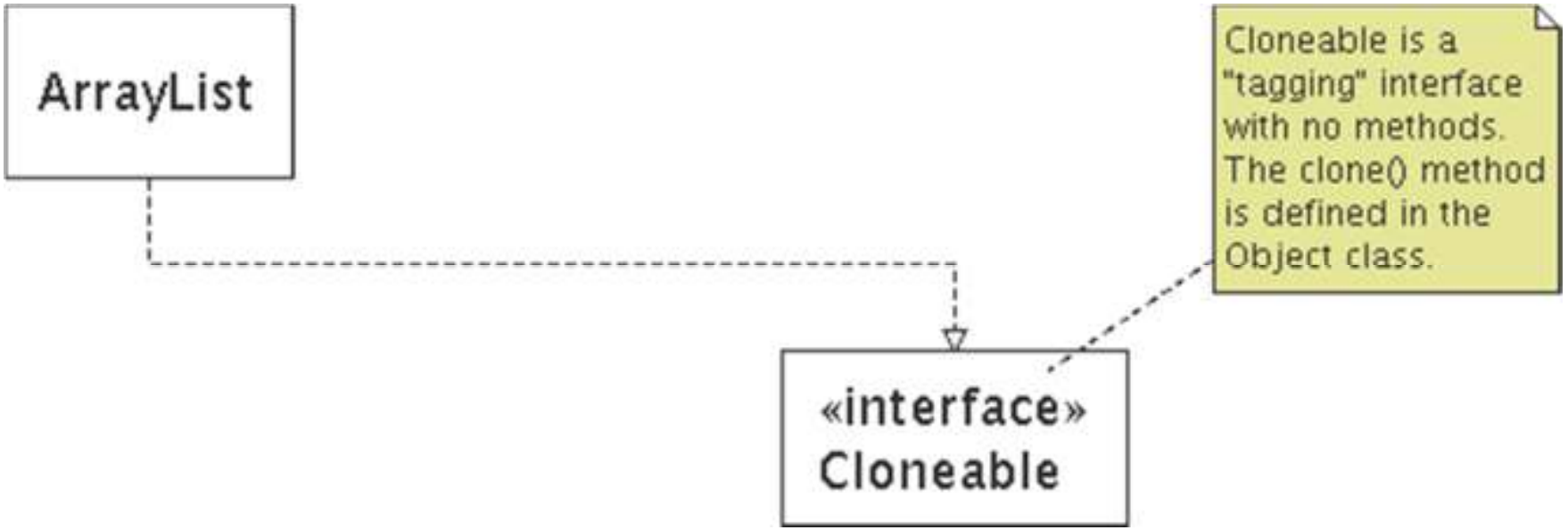
- operations / methods
  - *visibility name* (*parameters*) : *return_type*
  - visibility: + public
    # protected
    - private
    ~ package (default)
  - underline static methods
  - parameter types listed as (name: type)
  - omit *return_type* on constructors and when return type is void
  - method example:
    + distance(p1: Point, p2: Point): double





Comments

- represented as a folded note, attached to the appropriate class/method/etc by a dashed line



Relationships between classes

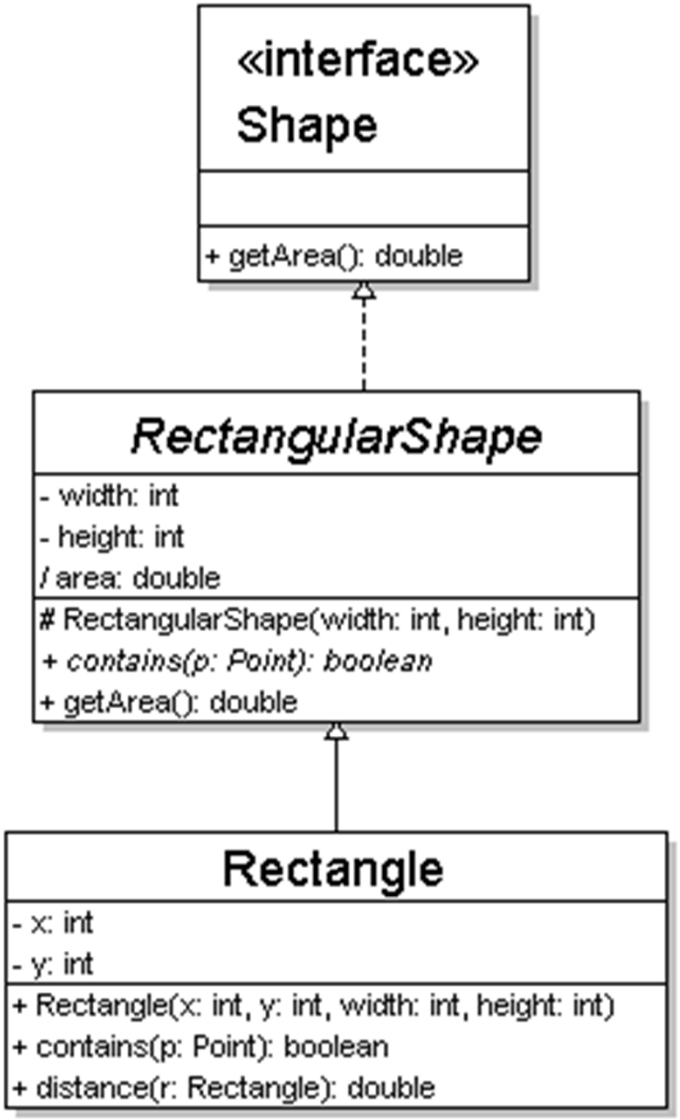
- **generalization**: an inheritance relationship
  - inheritance between classes
  - interface implementation

- **association**: a usage relationship
  - dependency
  - aggregation
  - composition

Generalization (inheritance) relationships

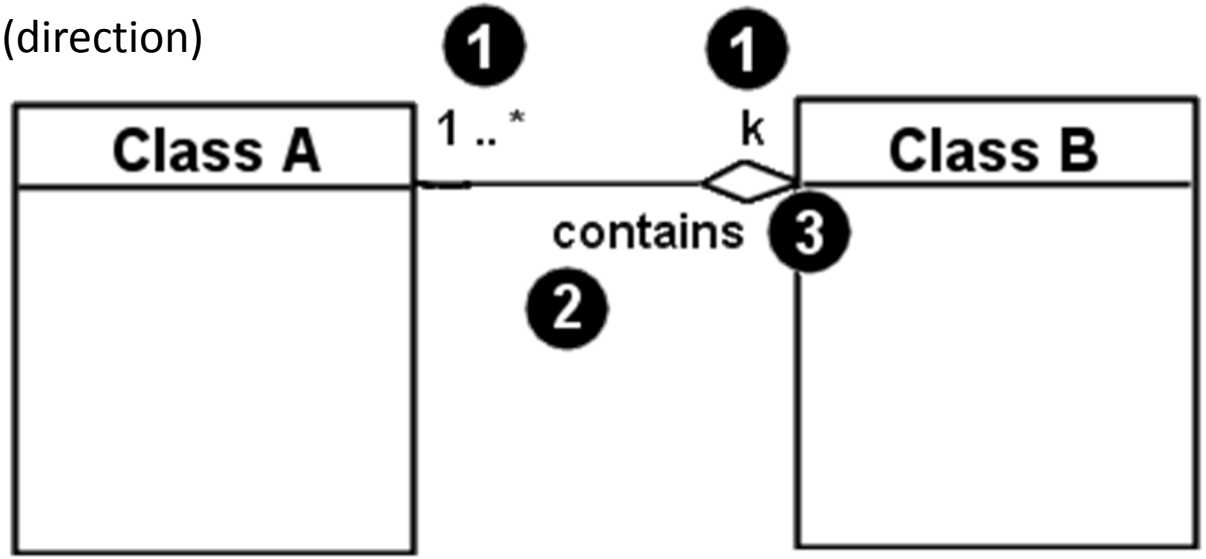
- hierarchies drawn top-down
- arrows point upward to parent
- line/arrow styles indicate whether parent is a(n):
  - class:
    solid line, black arrow
  - abstract class:
    solid line, white arrow
  - interface:
    dashed line, white arrow

- often omit trivial / obvious generalization relationships, such as drawing the Object class as a parent



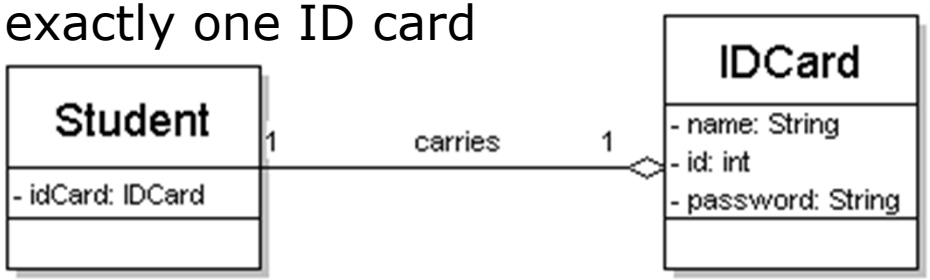
Associational relationships

- associational (usage) relationships
  1. multiplicity (how many are used)
     - * ⇒ 0, 1, or more
     - 1 ⇒ 1 exactly
     - 2..4 ⇒ between 2 and 4, inclusive
     - 3..* ⇒ 3 or more (also written as “3..”)
  2. name (what relationship the objects have)
  3. navigability (direction)

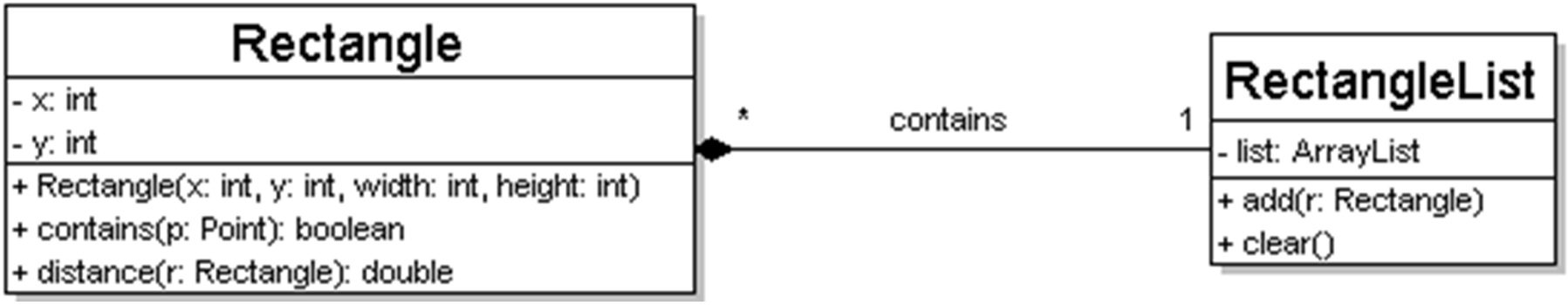


Multiplicity of associations

- one-to-one
  - each student must carry exactly one ID card

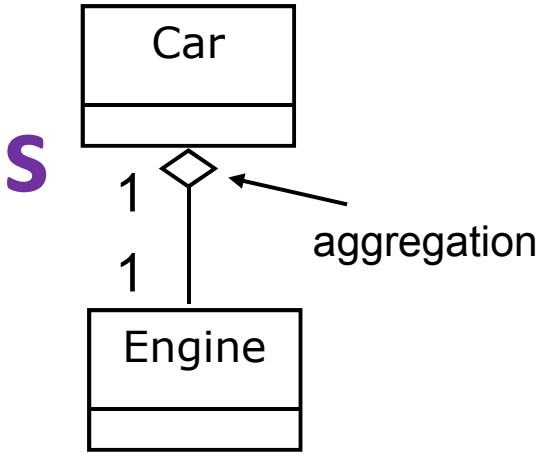


- one-to-many
  - one rectangle list can contain many rectangles

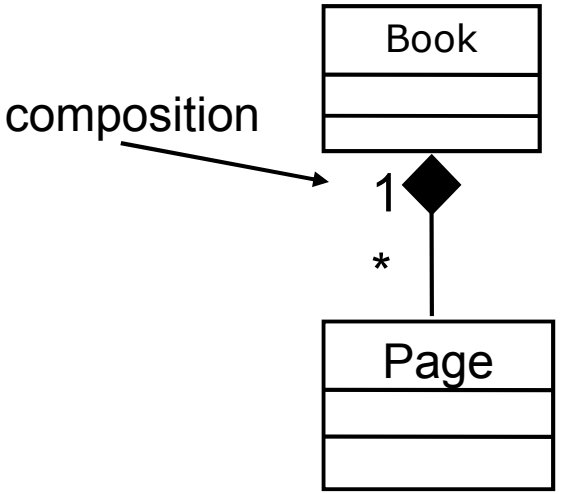


Association types

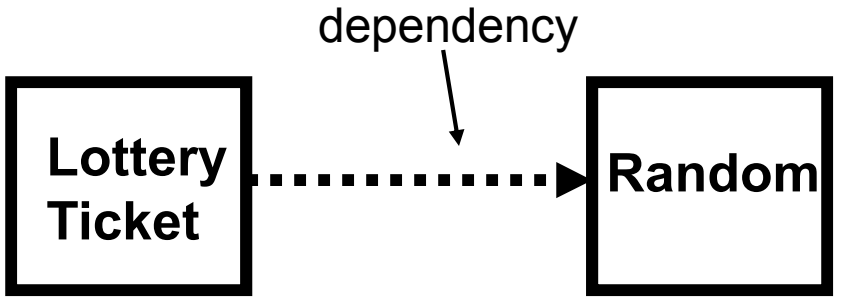
- **aggregation**: "is part of"
  - symbolized by a clear white diamond



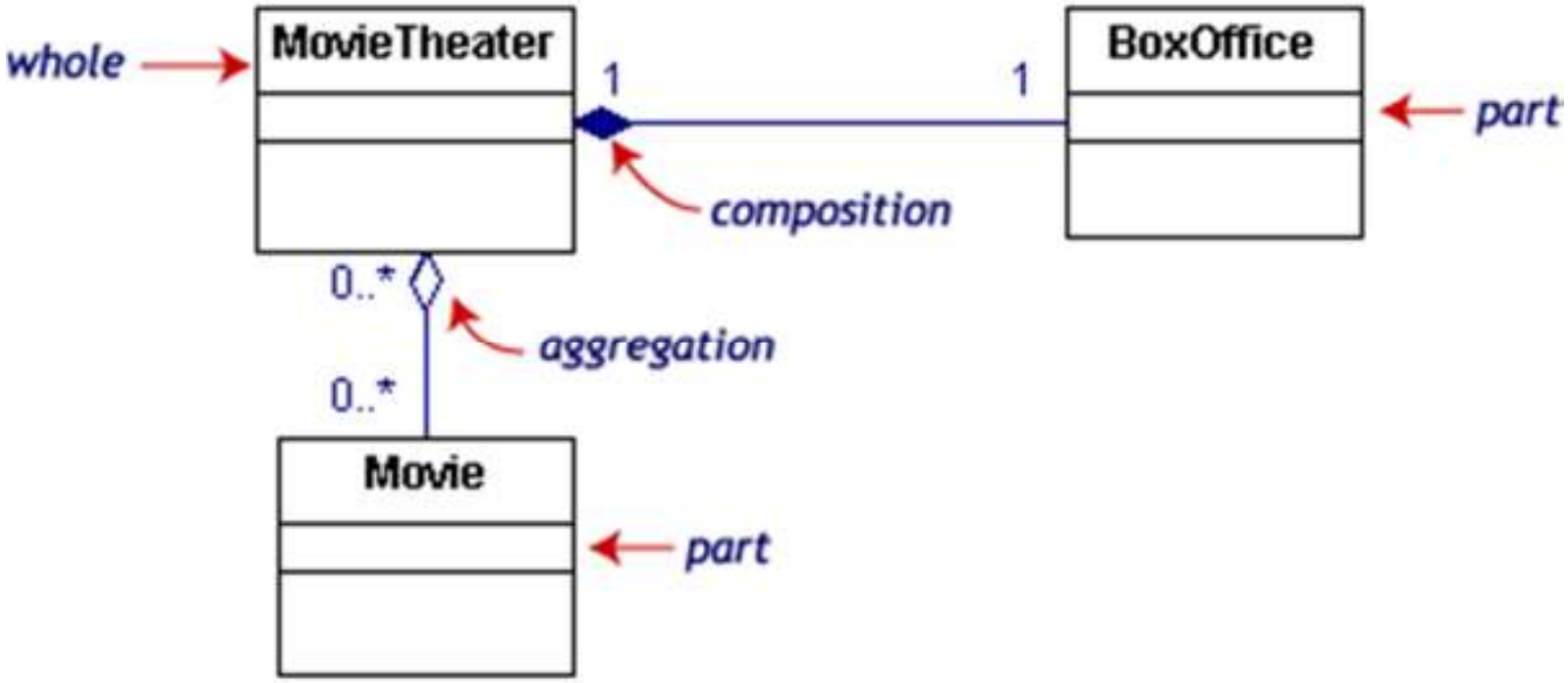
- **composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond



- **dependency**: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of that object's state



Composition/aggregation example

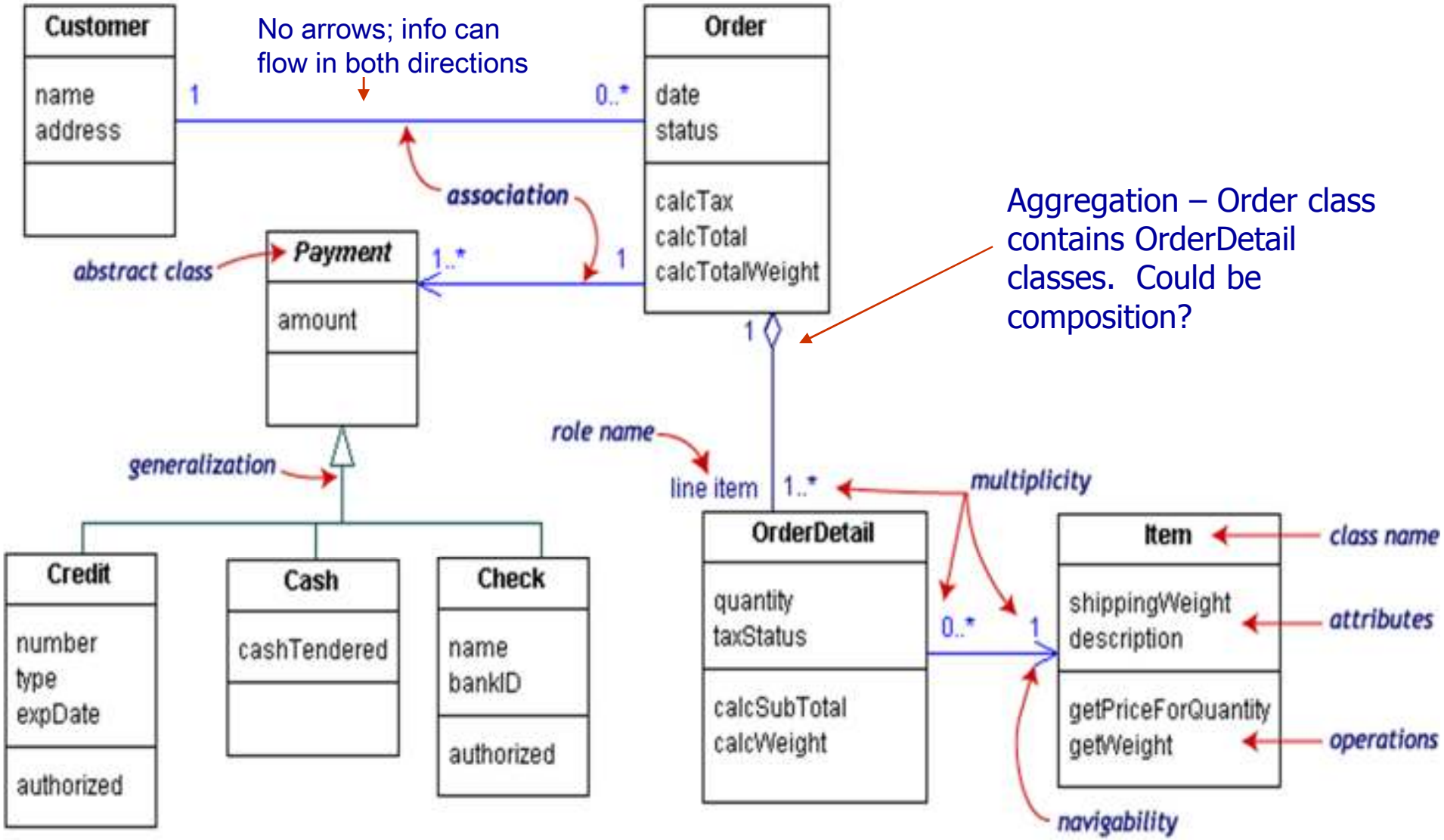


If the movie theater goes away

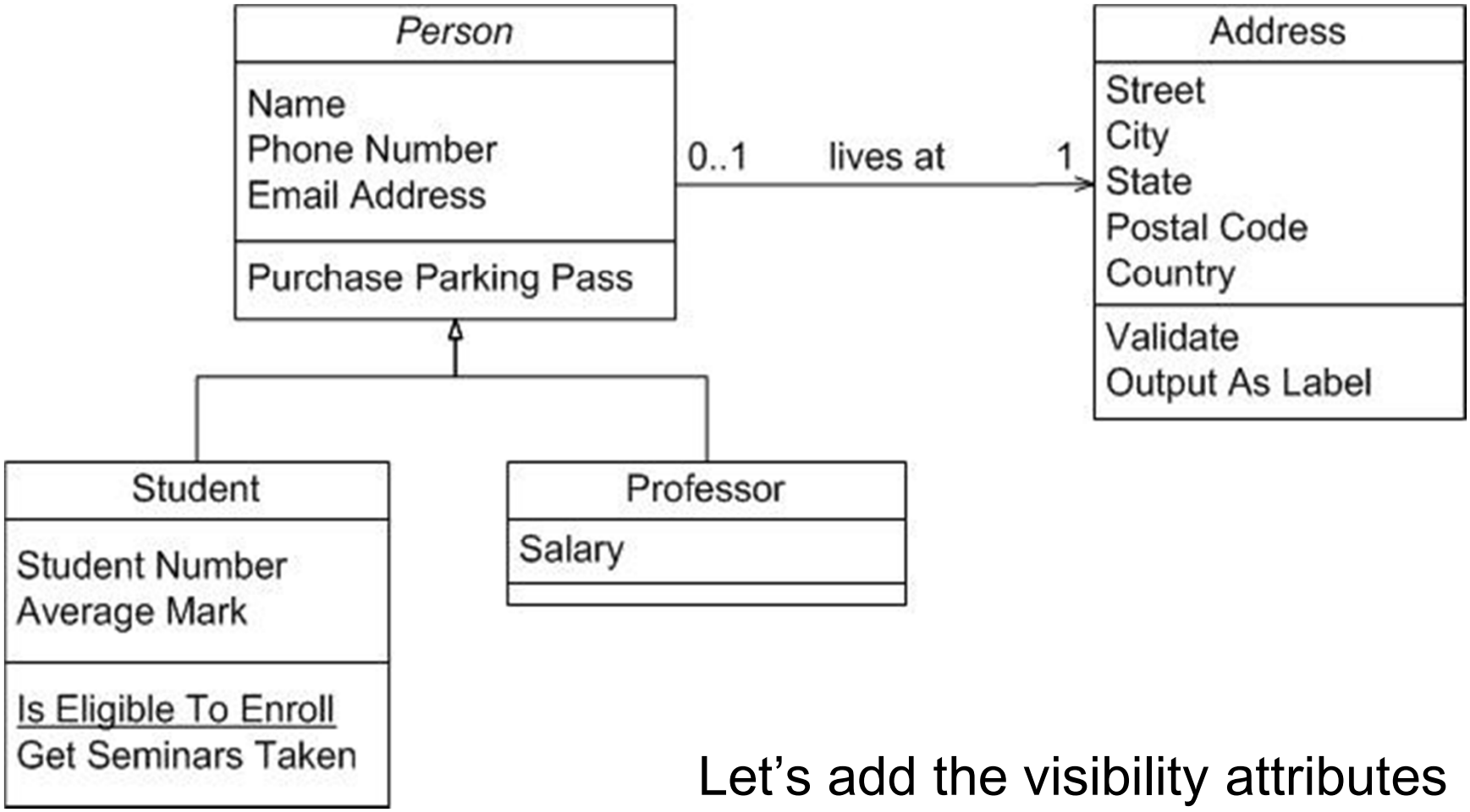
so does the box office => composition

but movies may still exist => aggregation

Class diagram example

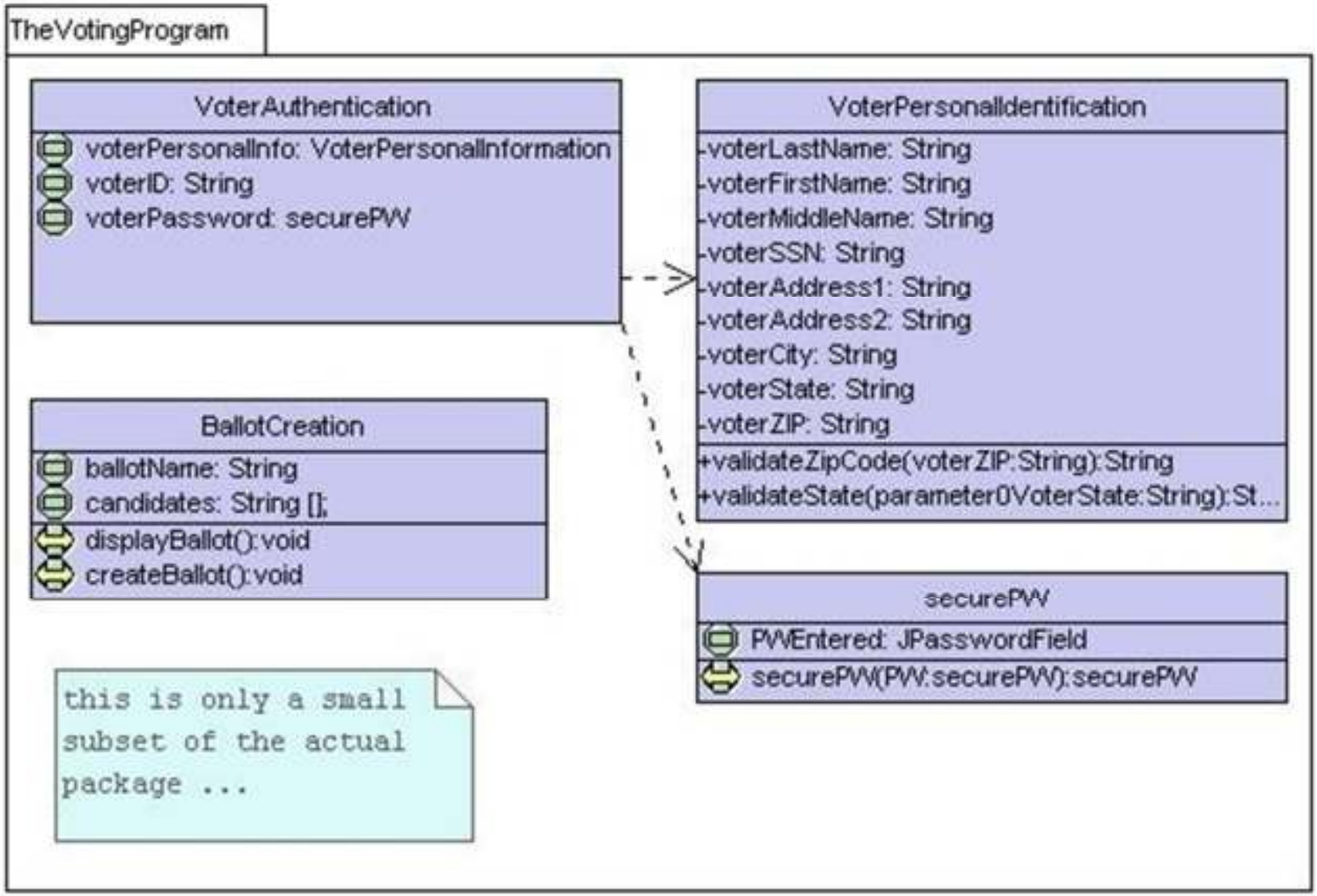


UML example: people

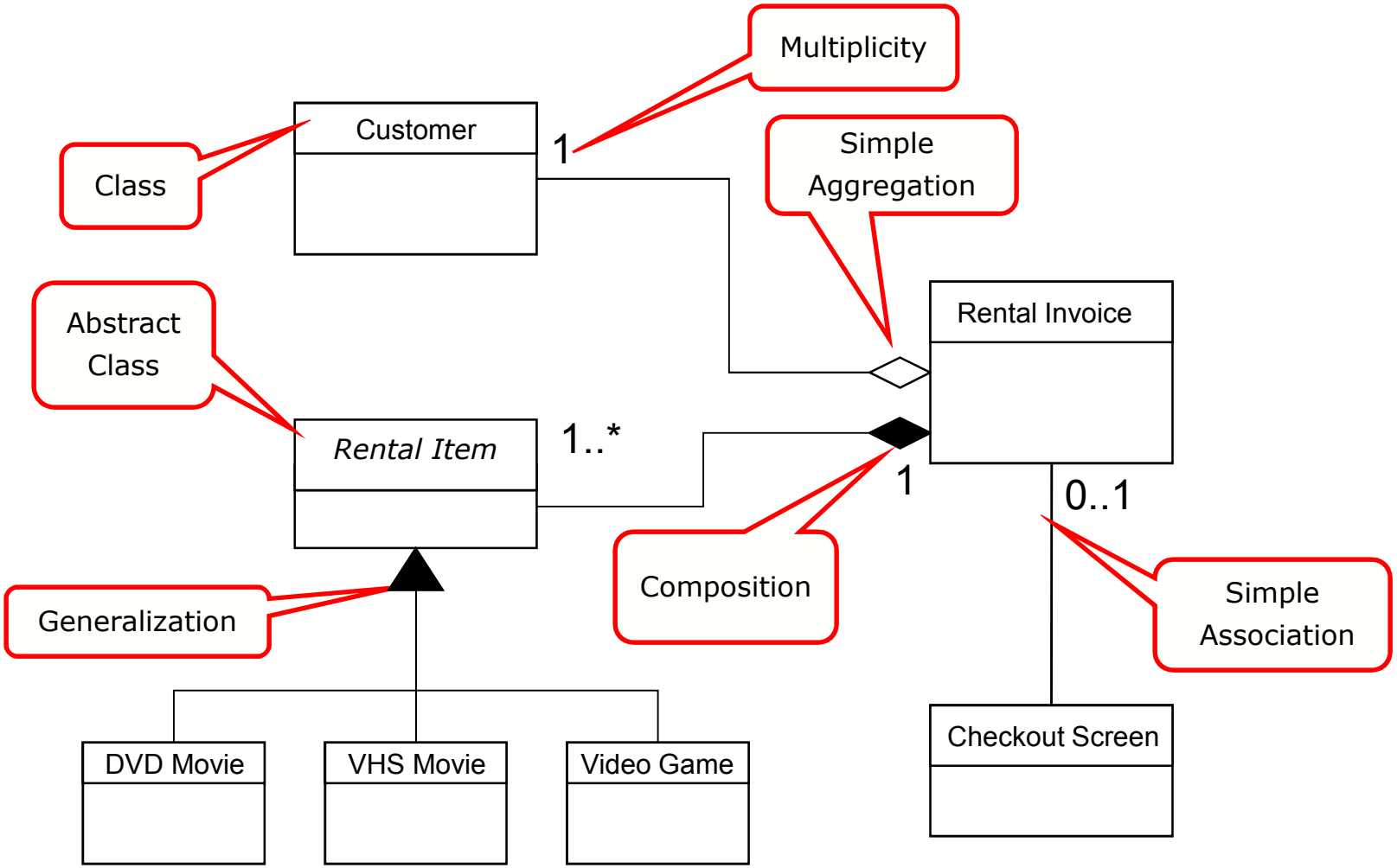


Let's add the visibility attributes

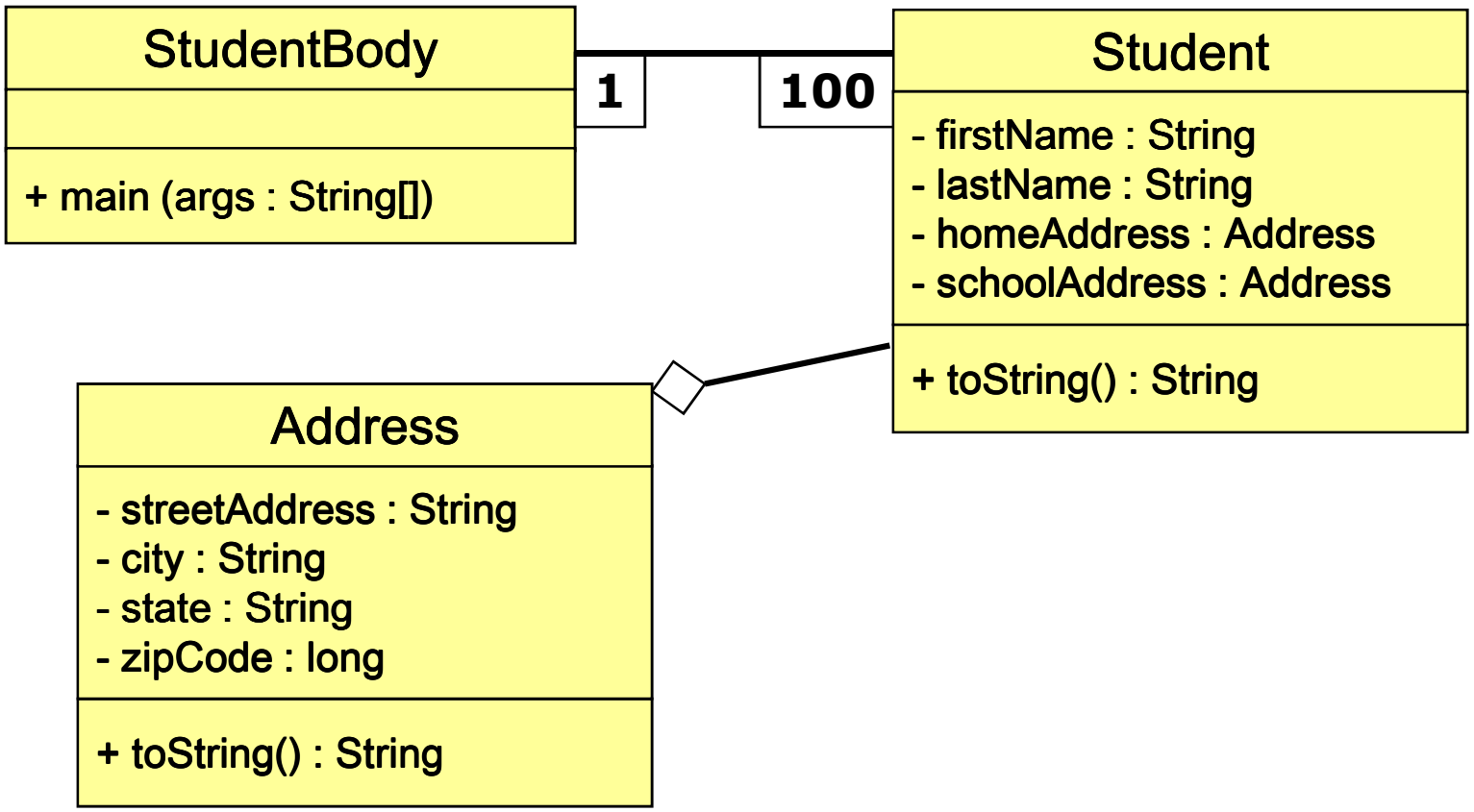
Class diagram: voters



Class diagram example: video store



Class diagram example: student



Tools for creating UML diagrams

- Violet (free)
  - http://horstmann.com/violet/
- Rational Rose
  - http://www.rational.com/
- Visual Paradigm UML Suite (trial)
  - http://www.visual-paradigm.com/
  - (nearly) direct download link:
    http://www.visual-paradigm.com/vp/download.jsp?product=vpuml&edition=ce

(there are many others, but most are commercial)

Design exercise: Texas Hold ‘em poker game

- 2 to 8 human or computer players
- Each player has a name and stack of chips
- Computer players have a difficulty setting: easy, medium, hard
- Summary of each hand:
  - Dealer collects ante from appropriate players, shuffles the deck, and deals each player a hand of 2 cards from the deck.
  - A betting round occurs, followed by dealing 3 shared cards from the deck.
  - As shared cards are dealt, more betting rounds occur, where each player can fold, check, or raise.
  - At the end of a round, if more than one player is remaining, players' hands are compared, and the best hand wins the pot of all chips bet so far.
- What classes are in this system? What are their responsibilities? Which classes collaborate?
- Draw a class diagram for this system. Include relationships between classes (generalization and associational).

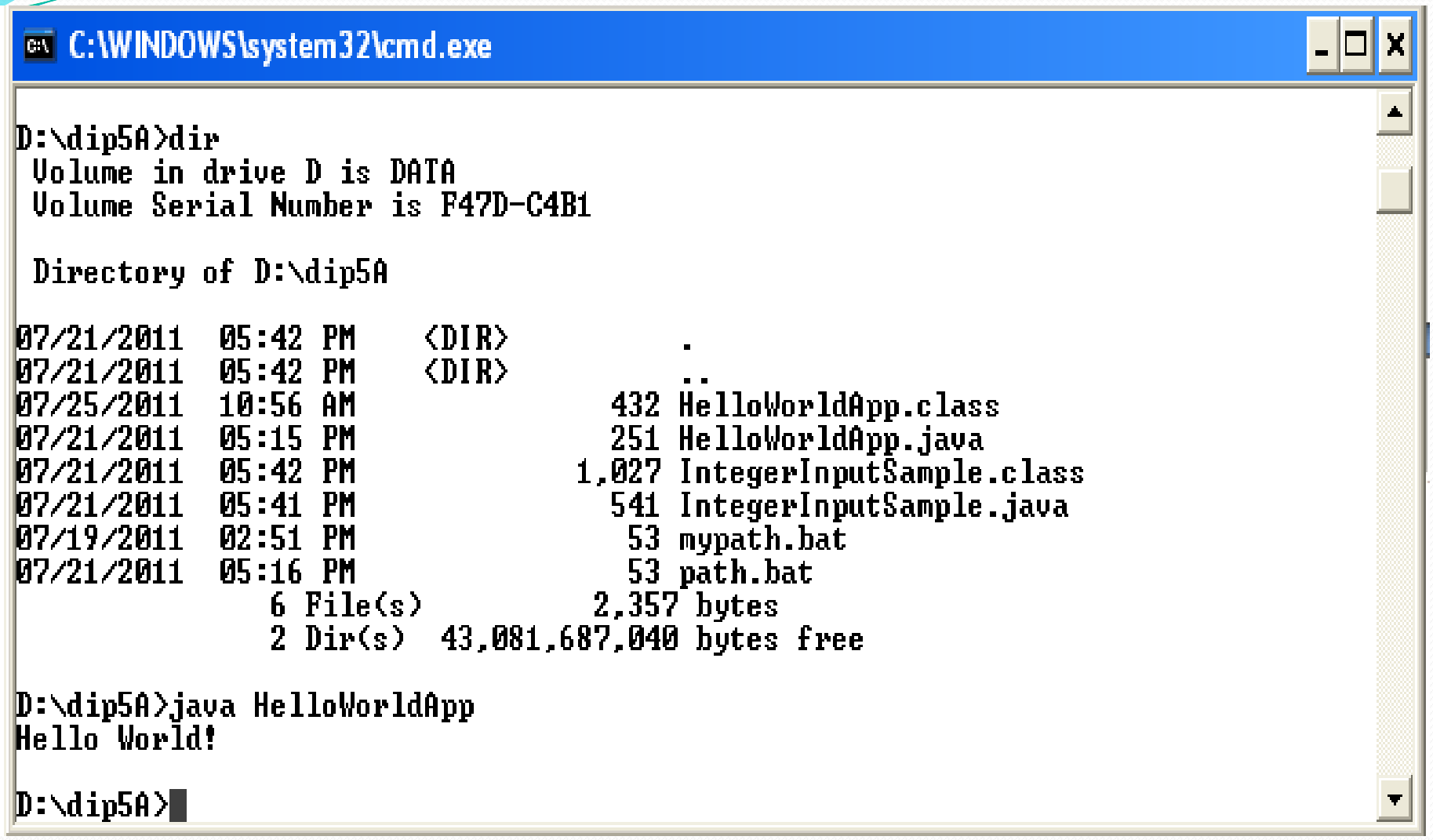
Class diagram pros/cons

- Class diagrams are great for:
  - discovering related data and attributes
  - getting a quick picture of the important entities in a system
  - seeing whether you have too few/many classes
  - seeing whether the relationships between objects are too complex, too many in number, simple enough, etc.
  - spotting dependencies between one class/object and another

- Not so great for:
  - discovering algorithmic (not data-driven) behavior
  - finding the flow of steps for objects to solve a given problem
  - understanding the app's overall control flow (event-driven? web-based? sequential? etc.)

CHAPTER 2

FUNDAMENTALS OF THE JAVA PROGRAMMING LANGUAGE

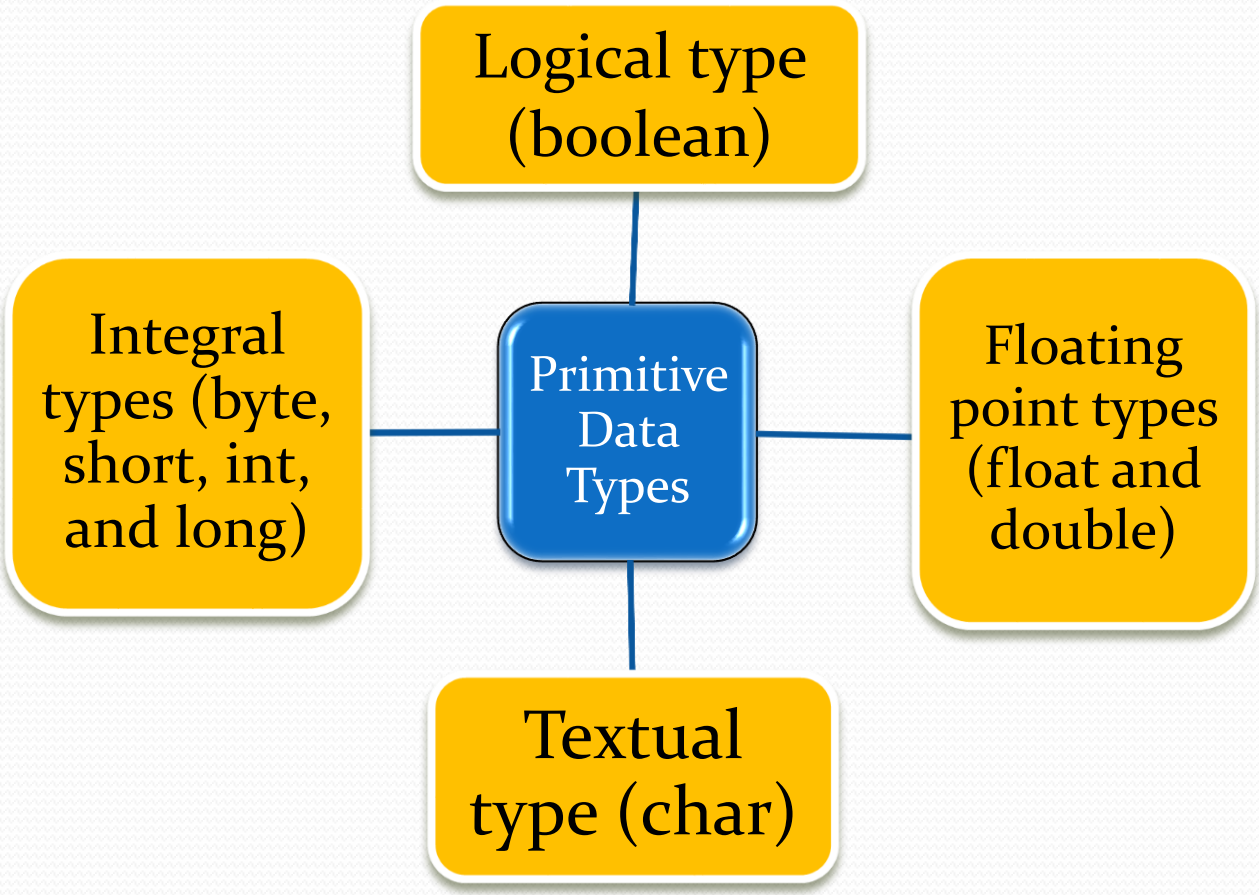

C:\WINDOWS\system32\cmd.exe
D:\dip5A>dir
Volume in drive D is DATA
Volume Serial Number is F47D-C4B1
Directory of D:\dip5A
07/21/2011 05:42 PM <DIR> .
07/21/2011 05:42 PM <DIR> ..
07/25/2011 10:56 AM 432 HelloWorldApp.class
07/21/2011 05:15 PM 251 HelloWorldApp.java
07/21/2011 05:42 PM 1,027 IntegerInputSample.class
07/21/2011 05:41 PM 541 IntegerInputSample.java
07/19/2011 02:51 PM 53 mypath.bat
07/21/2011 05:16 PM 53 path.bat
6 File(s) 2,357 bytes
2 Dir(s) 43,081,687,040 bytes free
D:\dip5A>java HelloWorldApp
Hello World!
D:\dip5A>

Lesson Learning Outcome

2.2 USE VARIABLES, OPERATORS AND INPUT/OUTPUT SREAMS.

- Declare, initialize, and use variables and constants according to Java programming language.
- Mathematical, relational and conditional operators in Java Programming.
- Type casting in Java programs.
- Implement type casting to change the data types.

Primitive Data Types



Integral Primitive Types

Type	Length	Range	Examples of Allowed Literal Values
`byte`	8 bits	-2^7 to 2^7 -1 (-128 to 127, or 256 possible values)	2 -114
`short`	16 bits	-2^{15} to 2^{15} -1 (-32,768 to 32,767, or 65,535 possible values)	2 -32699
`int`	32 bits	-2^{31} to 2^{31} -1 (-2,147,483,648 to 2,147,483,647 or 4,294,967,296 possible values)	2 147334778
`long`	64 bits	-2^{63} to 2^{63} -1 (-9,223,372,036854,775,808 to 9,223,372,036854,775,807, or 18,446,744,073,709,551,616 possible values)	2 -2036854775808L 1L

Floating Point Primitive Types

Type	Float Length	Examples of Allowed Literal Values
`float`	32 bits	99F -327456,99.01F 4.2E6F (engineering notation for $4.2 * 10^6$)
`double`	64 bits	-1111 2.1E12 999701327456 99.999

public double price = 0.0; // Default price for all shirts

Textual Primitive Type

- The only data type is **char**
- Used for a single character (16 bits)
- Example:
- public char colorCode = 'U';

Logical Primitive Type

- The only data type is **boolean**
- Can store only **true or false**
- Holds the result of an expression that evaluates to either true or false

Assigning a Value to a Variable

Example:

double price = 12.99;

Example (boolean):

- boolean isOpen = false;

Declaring and Initializing Several Variables in One Line of Code

- Syntax:

type identifier = *value* [, *identifier* = *value*];

- Example:

double price = 0.0, wholesalePrice = 0.0;

AdditionalWays to Declare Variables and Assign Values to Variables

- Assigning literal values:

```
int ID = 0;
float pi = 3.14F;
char myChar = 'G';
boolean isOpen = false;
```

- Assigning the value of one variable to another variable:

```
int ID = 0;
int saleID = ID;
```

- Assigning the result of an expression to integral,
  floating point, or Boolean variables

```
float numberOrdered = 908.5F;
float casePrice = 19.99F;
float price = (casePrice * numberOrdered);

int hour = 12;
boolean isOpen = (hour > 8);
```

CONSTANT

- Variable (can change):

  double salesTax = 6.25;

- Constant (cannot change):

  final double SALES_TAX = 6.25;

** Guideline – Constants should be capitalized with words separated by an underscore (_).

Source code example:

```
1   public class Shirt {
2
3   public int shirtID = 0; // Default ID for the shirt
4   public String description = "-description required-"; // default
5   // The color codes are R=Red, B=Blue, G=Green, U=Unset
6   public char colorCode = ’U’;
7   public double price = 0.0; // Default price for all shirts
8   public int quantityInStock = 0; // Default quantity for all shirts
9
10   // This method displays the values for an item
11   public void displayInformation() {
12   System.out.println("Shirt ID: " + shirtID);
13   System.out.println("Shirt description:" + description);
14   System.out.println("Color Code: " + colorCode);
15   System.out.println("Shirt price: " + price);
16   System.out.println("Quantity in stock: " + quantityInStock);
17
18   } // end of display method
19   } // end of class
```

Operator

Operator

- Is a symbol that instructs Java compiler to perform an operation, or action.

Mathematical Operators

- Are operators that are used for performing simple mathematical calculations such as addition, subtraction, multiplication and division.

- Write the resultant value of the following expressions.

Expression	Result
14-4	
14+4	
14* 4	
14/4	
14%4	

Assignment Operator

Example 1

x = 3;

Syntax

<Variable name> = <value>;

Example 2

$x = x + 3; \rightarrow x \mathrel{+}= 3;$

Syntax

<Variable name> <operator>= <value>;

Increment Operator

- Increments the existing value by one.

Example

```
num++;
```

Decrement Operator

- Decrements the existing value by one.

Example

```
num--;
```

Relational Operator

- Are used to perform a logical comparison of values, which results in a value that is either true or false.

Example

$x = 10;$

$x < 20;$

Relational Operator

Operator	Description	Example
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <=b
!=	Not equal to	a != b
==	Equal to	a == b

Relational Operator

Expression	Result
15 < 20.75	True
15 > 20.75	False
15 == 15	True
15 <= 15	True
15 >=20.75	False
15 != 15	True

Try to answer……

Expression	Result
4.5 <= 10	
4.5 < -10	
10 < 7+5	
-35 >= 0	

Logical Operator

- Are used to combine two or more logical expressions to form a more complex logical expression.

Example

(number<0 || number>100)

Logical Operator

Operators	Description
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical Operator

Condition1	Condition2	&&	||
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Conditional Operator

- Is used to construct conditional expressions.

Example

```
int aggregate = 95;
char grade = (aggregate>90)?'A': 'B';
```

Operator Precedence

- Determines the order in which mathematical computations and operations are performed in an expression.

Operator Precedence

Operator	Description
( )	Parenthesis
++, --	Increment/Decrement Operator
*, /, %	Mathematical Operator
+, -	Mathematical Operator
<, <=, >, >=, >	Relational Operator
= =, !=	Relational Operator
&&, ||	Logical Operator
?:	Conditional Operator
=	Assignment Operator

Operator Precedence

Result = a + b * 2 / c – 5 \\a=b=5, c=2

Result = 5 + 5 * 2 / 2 – 5

Result = 5 + 10 / 2 – 5

Result = 5 + 5 – 5

Result = 10 – 5

Result = 5.

Expressions

- Is any combination of operators and operands that evaluates to a value.
- The expressions can be classified into three main categories:
  - Numerical Expressions - Combine variables, numbers and mathematical operators.
  - Assignment Expressions - Assign values to variables.
  - Logical Expression - Results in true or false.

What is the answer????

S. No.	a	b	c	d	Expression	Result
1	24	2	8	-	a/b-c	
2	4	2	5	-	a-b+c	
3	2	4	2	1	a+b/c+d	
4	½	3.5	2.0	-	(a+b)*c	
5	-34	6	-	-	a/b	
6	-34	6	-	-	a%b	

Take


TYPECASTING
SIMILAR
MEDIA
PHOTOBLOGS
TYPEWRITTEN
PROCESSORS
UNEDITED
BLOGGING
MEANS
PUBLISHING
APPEAL
NOTES
ACTION
SIGNS
VANISHINGTEXT
ENTERED
DOMAIN
SERVICES
HANDWRITTEN
DIFFERENTIATED
COMPUTER
SKILL
PORTIONS
FORM
SOUND
FORMAT
TOOLS
BLOG
RAW
PAY
TYPECAST
STYLES
SENTENCE

Typecasting

- Is the conversion of a value of one data type into another data type.
- Typecasting is of two types:
  - Implicit conversion
  - Explicit conversion

Implicit Conversion

- When a value of data type with lower range is assigned to a variable of data type with higher range, Java automatically makes the conversion.

```
class ImplicitConversionSample
{
    public static void main(String args[])
    {
          double x; // Size is 8 byte

         int y=2; //Size is 4 bytes
          float z=2.2f;
          x=y+z;//Implicit conversion
         System.out.println("Result "+x);
    }
}
```

Explicit Conversion

- When a value of data type with higher range is assigned to a variable of data type with lower range, the user needs to make explicit conversion.
- Explicit conversion leads to data loss.

```
class ExplicitConversionSample
{
        public static void main(String args[])
        {
           int total_value;
           double book1 = 300.40;
           total_value=(int)book1; // Explicit conversion
           System.out.println("The total value is:"+total_value);

        }
}
```

Compile and Execute Java Program

-Practical Session-

Activity 1

Write a Java program to declare a variable d of type double. Assign the value 102.5 to d. Declare a float variable fl. Assign the value of d to fl by performing an explicit conversion and print the value on the screen.

Debugging Tips

- Error messages state the **line number** where the error occurs. That line might not always be the actual source of the error.
- Be sure that you have a **semicolon** at the end of every line where one is required, and no others.
- Be sure that you have an **even number of braces**.
- Be sure that you have **used consistent indentation** in your program.

Next lecture…

- Define input stream (System.in) and output stream (System.out) in Java programs.
- Implement input stream (System.in) and output stream (System.out) in Java programs.
- Write program using input and output stream.

Programming Language (JAVA)

METHOD

LESSON LEARNING OUTCOME:

At the end of this presentation, you will be able to:

- Define method in Java programs.
- Identify the use of methods in Java programs.
- Write methods in Java programs.

METHOD DEFINITION

- Is a component of the program written to perform a specific task.
- A program can be split into number of smaller programs called modules.

ADVANTAGES OF USING METHOD

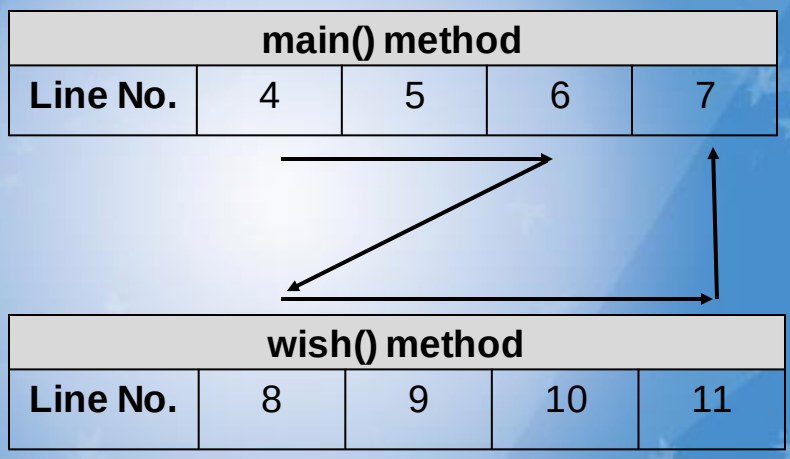
- The code written once can be used many times anywhere in the program.
- As the program is segregated into different smaller modules, locating the errors and correcting it is very easy.

HANDS-ON!

- Program Handson1.java will illustrate how to use method in java.

```
1.  class Handson1
2.  {
3.
4.      public static void main(String args[])
5.      {
6.              wish(); //calling method
7.      }

8.      static void wish()  // Method definition
9.      {
10.             System.out.println("Good  Morning");
11.     }
12. }
```

HANDS-ON!



DEFINING A METHOD

Syntax:

<return_type> <method_name> (<data_type1> <var1>, <data_type2> <var2>……)

```
{
    Statements;
}
```

Example:

```
int area(int radius, double pi)
{
    statements;
}
```

Example : Comparison Of Void Method And Double Method In Java Application Function

Example 1:

modifier

void, method that will not return a value

Data sent to the method

```
public void display(int num)
{   while(num<=5)
    {       System.out.print("*");
      num++;
    }
}
```

Example 2:

Method returns double type value

Method name

Data sent to the method

```
double count(int x, int y)
{   int sum = x + y ;
    return sum ;
}
```

count() method returns sum value

CALLING A METHOD

Example 1

area(4.5);

Syntax

method_name (var1, var2, …..);

CALLING A METHOD

Example 2

obj1.area(4);

Syntax

Object_name.function_name (var1, var2, …..);

HANDS-ON!

- Program Handson2.java will illustrate how to call methods using objects in java .

```
class Handson2
{
        public static void main(String args[])
        {
                Add obj=new Add(); //create object/instance of Add class
                obj.a=5;
                obj.b=3;          //Initialize a value for instance variable a,b
                System.out.println("The sum is " + obj.sum());
                                                        //calling a method from Add
        }                                               //class using object/instance obj.
}

class Add
{
        int a,b;
        int sum() //method definition
        {
                return (a+b);
        }
}
```

PASSING PARAMETERS

- There are two ways of passing parameters to a method.
- They are:
  - Pass-by-value
  - Pass-by-reference

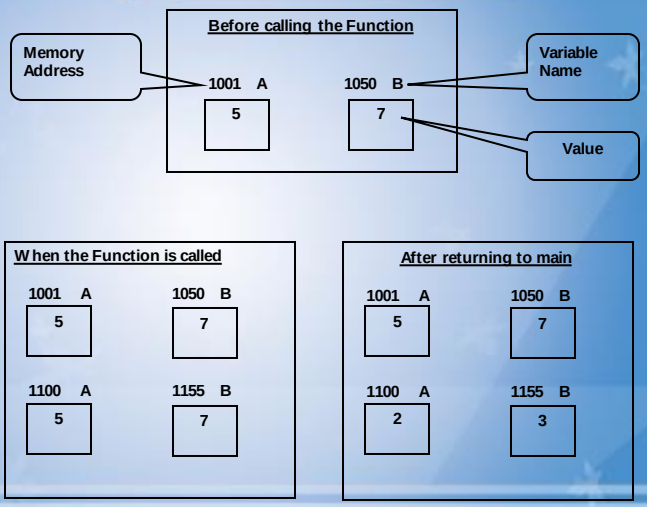
PASS BY VALUE

- When parameters are passed by value, **ONLY** the **VALUE** of the variable is passed to the method.
- Any change made to the variable in the called method will **NOT AFFECT** the variable in the calling method.

EXAMPLE : PASS PARAMETERS BY VALUE

```
class HandsOn2
{
        public static void main(String args[ ])
        {
                int A=5,B=7;
                System.out.println("Before calling the method:");
                System.out.println("A = " + A);
                System.out.println("B = " + B);
                change(A,B);
                System.out.println("After calling the method:");
                System.out.println("A = " + A);
                System.out.println("B = " + B);
                System.out.println("\nThis is pass by value");
        }

        static void change (int A,int B)
        {
                A=2;
                B=3;
        }
}
```

HANDS-ON!



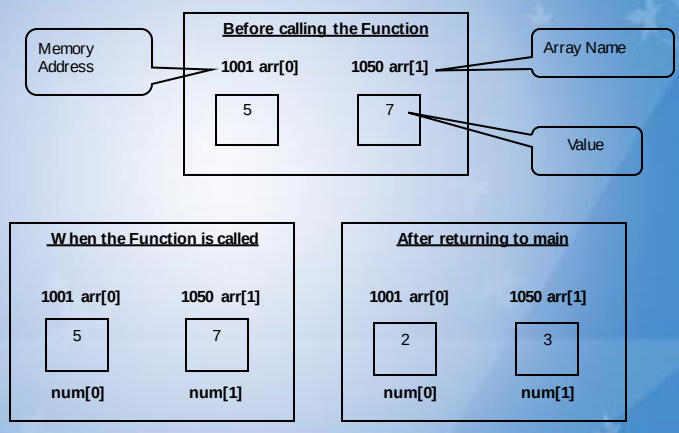
PASS BY REFERENCE

- When parameters are passed by reference, the **ADDRESS** of the variable is passed to the method.
- Any change made to this variable in the called method will **AFFECT** the variable in the calling method.

EXAMPLE : PASS PARAMETERS BY REFERENCE

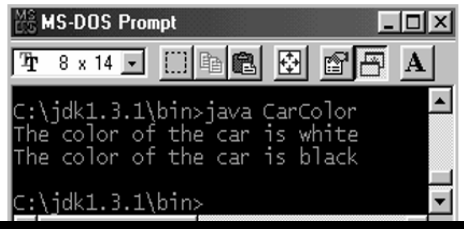
```
public class Reference
{
        public static void main(String args[])
        {
                int [ ] arr = {5,7};
                System.out.println("Before calling the method:");
                System.out.println("arr[0] = " + arr[0]);
                System.out.println("arr[1] = " + arr[1]);
                change(arr);
                System.out.println("After calling the method:");
                System.out.println("arr[0] = " + arr[0]);
                System.out.println("arr[1] = " + arr[1]);
                System.out.println("This is pass by reference");
        }

        static void change(int [ ] num)
        {
                num[0]=2;
                num[1]=3;
        }
}
```

HANDS-ON!



```
class CarColor
{
      public static void main(String arg[])
      {     Car Mini = new Car();
            System.out.println("The color of the car is " + Mini.color);
            paintCar(Mini, "black");
            System.out.println("The color of the car is " + Mini.color);
      }

      public static void paintCar(Car C2, String s)
      {
            C2.color = s;
      }
}


class Car
{ double cc=1000;
  String color = "white";
}
```

Program Output:



Pop Quiz !!!

QUESTION 1

1. Detect the errors in the following program and correct them:

```
class Activity1
{
    public static void main(String args[])
    {
        int n=Integer.parseInt(args[]);
        System.out.println("The cube of " + n + " is " + cube(n));
    }

    static float cube{int n}
    {
        return (n*n*n);
    }
}
```

QUESTION 2

2. Fill in the blanks in the program

```
// Program to check for leap year using boolean method
class Activity2
{
    public static void main(_______ args[])
    {
        _____ yr=Integer.parseInt(args[0]);
        if (leap(yr))
                System.out.println("Leap  year");
        else
                _______.out.println("Not a Leap year");
    }

    static ________ leap(____ yr)
    {
        if (yr % 400 == 0)
                return true;
        if (yr % 100 == 0)
                return false;
        if (yr % 4 == 0)
                return true;
        else
                return false;
    }
}
```

QUESTION 3

3. Add the method definition that returns your name in the following program:

```
class Activity3
 {
   public static void main(String args[])
   {
      System.out.println("My name is " + getName());
   }
   ____________________
   ____________________
   ____________________
   ____________________
 }
```

QUESTION 4

4. Add the statement to call the method *factorial()* in appropriate place:

```
class Activity4
{
      public static void main(String args[])
      {
            int n=Integer.parseInt(args[0]);
            System.out.println("The factorial of "+n+" is "+ _____)
       }

      static int factorial(int no)
      {
            int fact=1;
            for (int i=1;i<=no;i++)
            {
                  fact=fact*i;
            }
            return fact;
       }
}
```

SUMMARY

In this session, you learnt the following:

- A Method itself is a small program written to perform a specific task.
- The code written once can be used many times anywhere in the program (Code reusability) using methods.
- If a method is declared as private, this method can be called only by the methods of the same class.

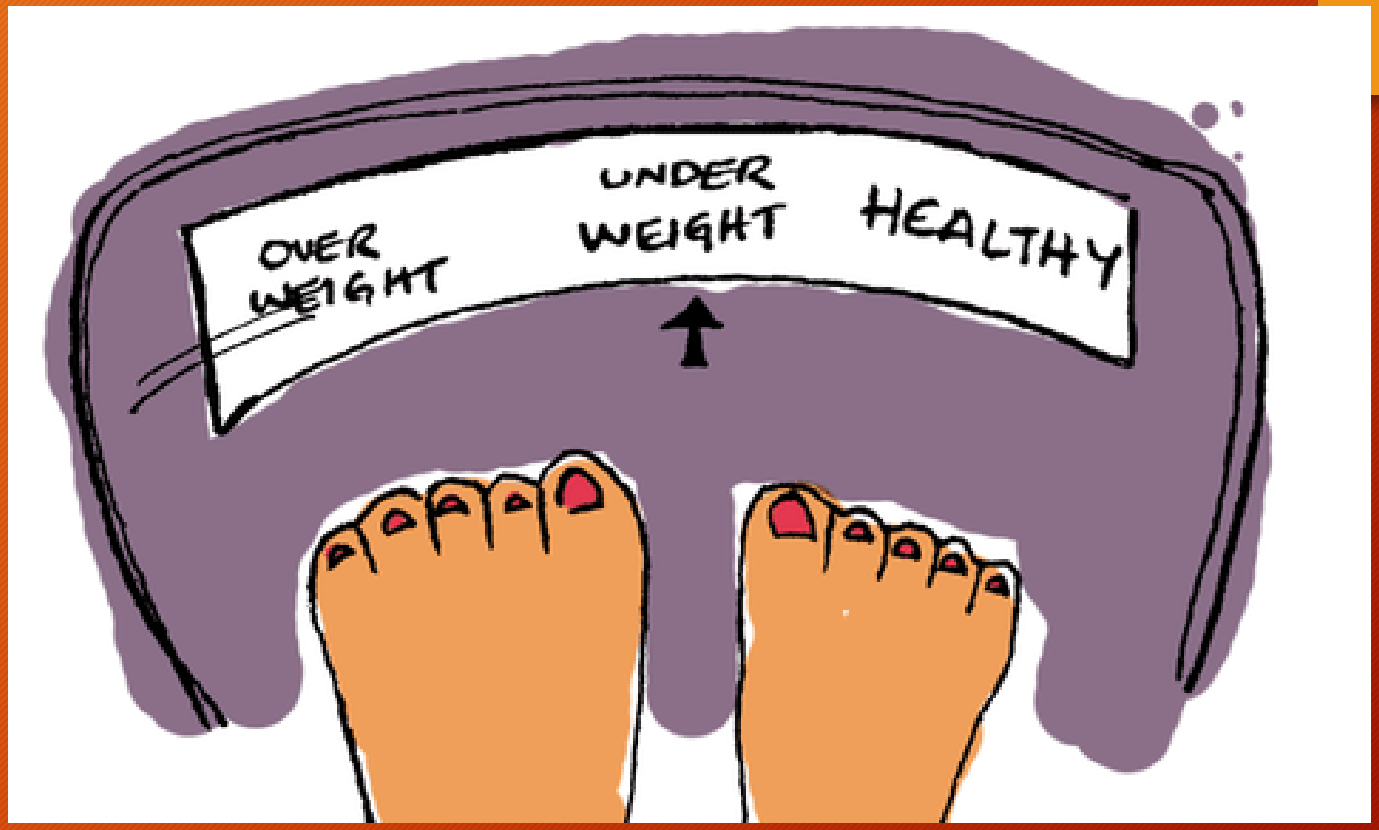
SUMMARY

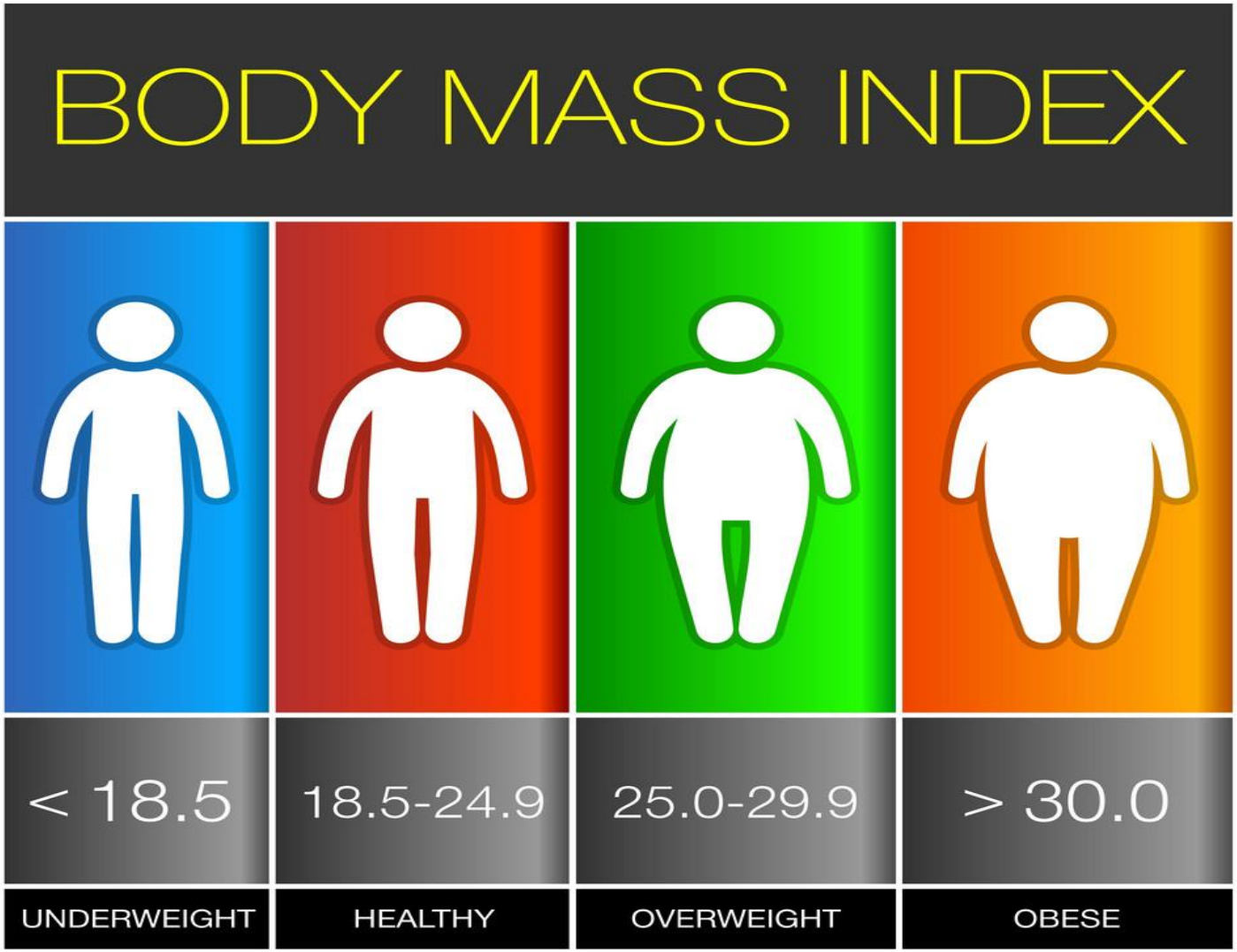
In this session, you learnt the following:

- Static methods can be accessed without creating objects.
- If a method does not return any value, void should be specified.
- Two ways of passing parameters are Pass-by-value and Pass-by-reference.

CHAPTER 3: (*continue*) CLASSES AND OBJECTS

3


OVER WEIGHT
UNDER WEIGHT
HEALTHY


BODY MASS INDEX
< 18.5
18.5-24.9
25.0-29.9
> 30.0
UNDERWEIGHT
HEALTHY
OVERWEIGHT
OBESE

Activity 1 : Review

Write a program that declare variables ***height*** and ***weight*** of **double** data type and assign the values **1.65m** (height) and **65kg**(weight).

Declare a **resultBMI** as **integer** data type and calculate the BMI using this formula:

resultBMI =(weight) ÷ (height x height)

Print the value of the variables weight, height and the BMI result.


"Success seems to be connected with action. Successful people keep moving. They make mistakes, but they don't quit."
-Conrad Hilton

Lesson Learning Outcome

Input and Output Streams.

- Use input and output statement in Java programs.
- Write java programs using Input/ Output Streams.

Input/Output Stream

- Input stream refers to the flow of data to a program from an input device (**System.in**).
- Output stream refers to the flow of data from a program to an output device **(System.out).**

3 Methods to get the input

- ✓Using **Buffered Reader** class
- ✓Using **Scanner** class
- ✓Using **Command line** approach

3 Methods to get the input

- ✓Using **Buffered Reader** class (5-6 steps)
  1) import file ->import java.io.*;
  2) ….throws IOException
  3) BufferedReader object
  4) Instruction (S.o.p)
  5) Retrieve the input (System.in)
  6) Convert into numeric (if necessary)

3 Methods to get the input

- ✓Using Scanner class (4 steps)
  1) import file ->import java.util.*;
  2) Scanner object
  3) Instruction (S.o.p)
  4) Retrieve the input (System.in)

3 Methods to get the input

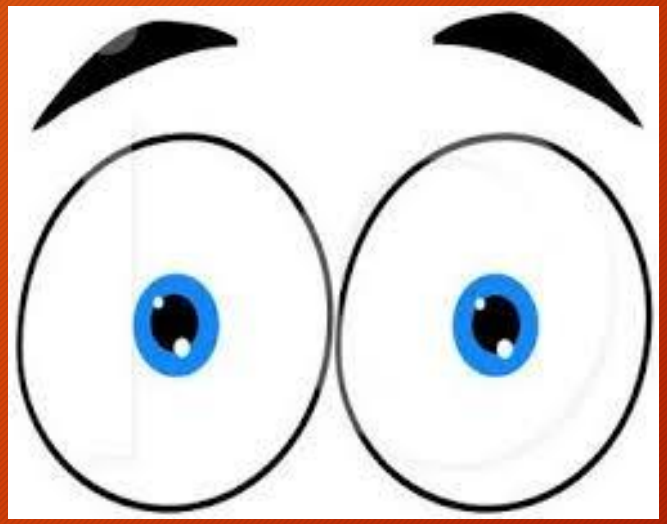
✓Using **Command line** approach

1) String temp = <array name>[index];
2) Convert into numeric (if necessary)

   int a = Integer.parseInt(temp)

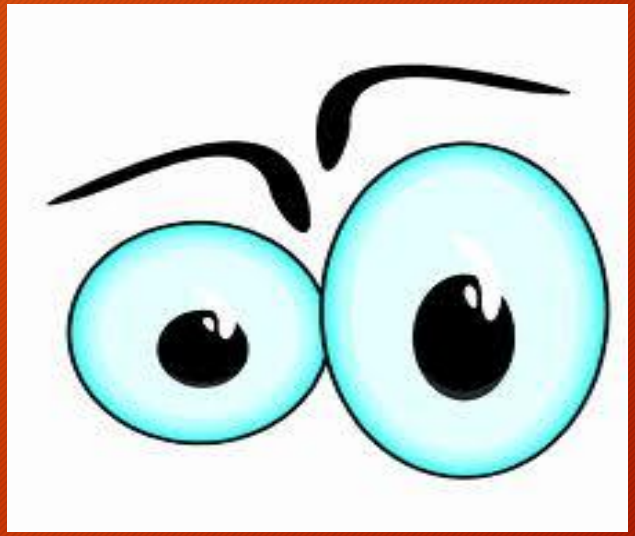
Hands-On!

- Program InputSample.java accepts input data using input stream.



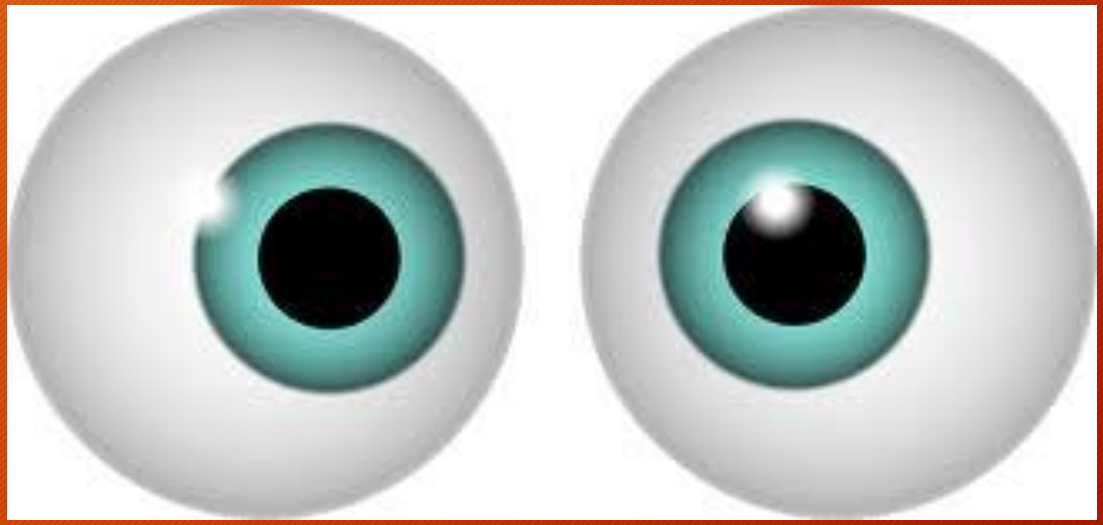
Hands-On!

- Program IntegerInputSample .java accepts input data using input stream and displays the value using output stream.



Hands-On!

- Program CommandLineInputSample .java accepts input from the command line.



Summary

In this session, you learnt the following

- In Java source of **input data** is called **Input stream** and source of **output data** is called **output stream**.
- All input and output is done by using methods that are part of **java.io package** in Java.
- Input can be provided to Java programs using command line arguments.

CHAPTER 3: (*continue*) CLASSES AND OBJECTS

3

Branching Statements

Lesson Learning Outcome

Use branching statements and arrays

- Explain selection statements in Java programs.
- Explain looping statements in Java programs
- Write program using branching statements.

Introduction

- Branching statements are used to **change the sequential flow of a program**.
- These **set of statements** are executed based on a condition.
- Java supports two types of branching statements:
  - **Selection** statement
  - **Looping** statement

Branching Statements

- Used to evaluate expressions and direct the program execution based on the result.
- The branching statements supported by Java are:
  - *if*
  - *switch*

Single if Statement

- Used to execute a set of statements when the given condition is satisfied.

Syntax:

```
if (<condition>)
{
  <Conditional statements>;
}
```

- Conditional statements within the block are executed when the condition in the *if* statement is satisfied.

Hands-On!

- Program InputValue.java illustrates the execution of a simple *if* statement. The program checks whether the given number is greater than *100*.

Hands-On!

- Program Marks.java illustrates the execution of a simple *if* statement based on two conditions. This program checks whether the marks is between *90* and *100* to print the Grade.

The if-else Statement

- Executes the set of statements in if block, when the given condition is satisfied.
- Executes the statements in the else block, when the condition is not satisfied.

Syntax:

```
if (<condition>)
{
   <Conditional statements1>;
}
else
{
   <Conditional statements2>;
}
```

Hands-On!

- Program sample.java illustrates the use of the *if-else* statement. This program accepts a number, checks whether it is less than *0* and displays an appropriate message.

Lab Exercise

2. Accept a number from the user and check whether it is divisible by 5.

   Hint: Use the modulus operator, %, to check the divisibility.

Lab Exercise

3. Accept a number from the user and check whether the number is an odd number or an even number.

   Hint: Use the modulus operator, %, to check the divisibility.

Nested if Statement

- The *if* statements written within the body of another *if* statement to test multiple conditions is called nested *if*.

Syntax :

```
if (<Condition 1>)
{
        if (<Condition 2>)
          {
            <Conditional statements1>;
          }
        else
          {
            <Conditional statements2>;
          }
}
else
{
        <Conditional statements3>;
}
```

Inner if condition

Outer if condition

Hands-On!

- The program highest.java illustrates the use of nested *if* statements. The program accepts three integers from the user and finds the highest of the three.

Lab Exercise

4. Input three numbers from the user and find the greatest among them.

Lab Exercise

5. During a special sale at a mall, a 5% discount is given on purchases above RM100.00. Write a program to accept the total purchase amount and calculate the discounted price.

Example

Enter the amount of purchase in RM: 2000
Discounted price = 2000-(2000*5/100) = 1900

The *switch* Statement

- Is a multi-way branching statement.
- Contains various case statements.
- The case statements are executed based on the value of the expression.
- A *break* statement passes the control outside switch structure.

Hands-On!

- Program SwitchDemo.java illustrates switch case execution. In the program, the *switch* takes an integer value as input and displays the month based on the integer entered.

Hands-On!

- The program SampleSwitch.java illustrates switch case execution. The program checks whether the given character is a vowel.

Lab Exercise

6. Write a program to accept the day of the week and print the day.

Example

1 – Sunday

5 – Thursday

Take

Looping Statements

- Used to execute a set of instructions repeatedly, as long as the specific condition is satisfied.
- The looping statements available in Java are:
  - *for*
  - *while*
  - *do-while*

The *for* Loop

- This looping statement enables you to repeat a set of instructions based on the condition.
- Used when the number of loops is known before the first loop.

Syntax

```
for(initialisation;condition; incrementation/decrementation)
{
    //loop statements
}
```

Exercise

- Write a progam to get 1 number (integer- from user) *-(inputStreams knowledge)*
  - Check either number is odd or even *-(selection statement knowledge)*
  - Get input from 20 users. *-(looping statement knowledge)*

The *for* Loop (Contd…)

- The for loop has three parts:
  - **Initialisation:** The initial value of the loop control variable is set here.
  - **Condition:** The loop condition is specified and the loop statements are executed as long as this condition is satisfied.
  - **Incrementation / Decrementation:** The initial value of the loop control variable is either incremented or decremented each time the loop gets executed.

Hands-On!

- The program LoopDemo.java illustrates the working of a *for* loop.

Activity

Step1: Open the data file LoopSample.java.

Step 2: Run and observe the output.

Step 3: Draw the appropriate flowchart for the program.

Lab Exercise

Write a program to print the following triangle of numbers:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

Lab Exercise

Write a program that will accept a number from the user and print the multiplication table for that number in the proper format.

Example

If the input is 2, print the 2 tables in the format:

2* 1=2

……..

……..

2* 10=20

Lab Exercise

9. Display the following series using *for* loop:

Series 1

1

1 2

1 2 3

1 2 3 4

Series 2

1, 3, 5, 7…n

Series 3

123

456

789

The *while* Loop

- Is a looping statement that enables you to repeat a set of instructions based on a condition.
- Used when the number of loops is not known before the first loop.

Syntax:

```
<Initialise variable>;
while(condition)
{
   //loop statements
  <Increment/decrement variable>;
 }
```

Hands-On!

- Program WhileDemo.java illustrates the working of a *while* loop.

Activity

Step 1: Open the data file oddno.java.

Step 2: Fill in the blanks in line 7 and 10 such that the output of the program is `1 3 5 7 9 11`

Step 3: Save, compile and run the program to observe the output.

Activity

Step 1: Open the data file SqNatNo.java.

Step 2: Modify the code line 8 to complete the condition to repeat the loop till it finds the square of first five natural numbers (1 to 5).

The *do-while* Loop

- The *do-while* loop is similar to the *while* loop.
- The difference only in the location where the condition is checked. In *do-while*, condition is checked at the end of loop.

Syntax:

```
do
  {
      //loop statements
  } while(condition);
```

Hands-On!

- Program doWhileDemo.java illustrates the working of a *do-while* loop.

Activity

Step 1:Open the data file count_1.java.

Step 2: Fill in the blanks with appropriate code

to increment the variable count.

Step 3: Save, compile and execute the code.

Activity

Step 1:Open the data file count_2.java

Step 2: Read the program and write the output.

Lab Exercise

11. Write a program that accepts an integer and print it backwards.

    For example,

    If the input is *53*

    Output*: 35*

Lab Exercise

12. Write a program that reads an integer and prints the sum.

    For example,

    If the input is *53*

    Output *8*

break and *continue*

- The *break* and *continue* statements in Java allow transfer of control from one statement to another.
- The ***break*** statement causes **termination** of the loop **and transfers** the control outside the loop.
- The ***continue*** statement will **transfer** the control to the beginning of the loop.

CHAPTER 3: (*continue*) CLASSES AND OBJECTS

3

Lesson Learning Outcome

- Define an array in java program.
- Create reference arrays of objects in Java program.
- Initialize elements of arrays.
- Pass and return array to method.
- Write program using single and multidimensional array.

Array Fundamentals

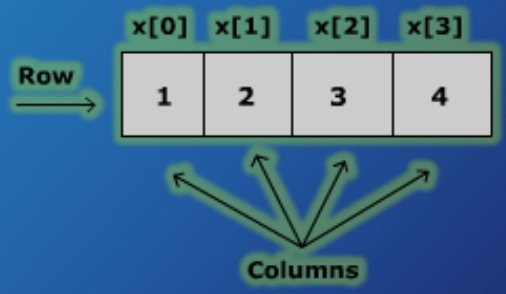
- An Array is a collection of similar type of variables having a common name.
- Values in array are stored in consecutive memory locations.
- Two types of Arrays
  - One-dimensional array
  - Two-dimensional array

One-Dimensional Array

- Will have a single row and can have any number of columns.
- Will have only one subscript. Subscript refers to the dimension of the array.

Example : int *x[ ] = new int[3];*

Representation of Array



Declaring and Initialising Arrays

- You need to declare an array before using it in the program.
- An array can be defined using different data types such as integer, double, float, char and so on.
- However, all the values in an array must be of the same data type.

Declaring an Array

Syntax:

<Data type> <Variable name>[ ] = new <Data type>[Size of the array];

Example: *int marks[ ] = new int[5];*

Initialisation of an Array

Initialisation is the process of assigning values to the array you have created.

Example :

```
marks[0] = 95;
marks[1] = 85;
marks[2] = 75;
marks[3] = 80;
marks[4] = 65;
```

Initialisation of an Array (method 1)

Syntax :

<Variable name>[Array index] = Value;

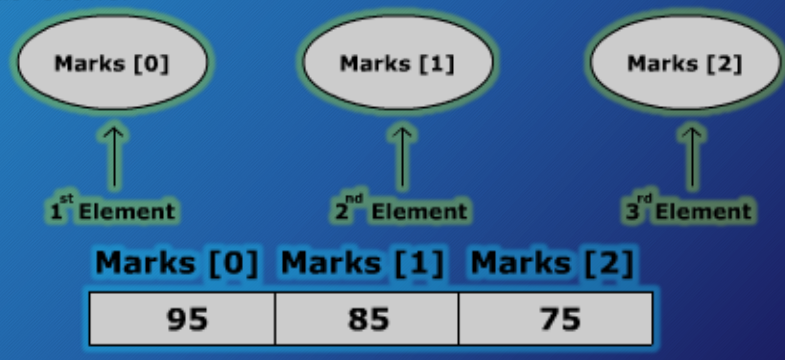
- The first element will always have the array index as 0.
- Array index refers to the location of the values in an array.

For example:

```
marks[0]=95;
marks[1]=85;
marks[2]=75;
```

Array Elements

The representation of the array for the example in the previous slide is as follows.



Here, 0, 1 and 2 are the index of the array, marks.

Initialise an Array (method 2)

You can also initialise the array at the time of declaration as shown in the following example:

```
int marks[] = {95,85,75};
```

When initialising an array, the values are enclosed within curly brackets **{ }**.

Accessing Array Elements

- Using the array index you can access the array elements.
- To print the value of the second element in an array, the code will be:

*System.out.println("The second element:" + **marks[1]**);*

Example:

```
class Student_Marks
{
  public static void main(String args[])
  {
          int marks[ ] = {95,85,75,80,65};
          System.out.print("\n marks[0]  : " + marks[0]);
          System.out.print("\n marks[1]  : " + marks[1]);
          System.out.print("\n marks[2]  : " + marks[2]);
          System.out.print("\n marks[3]  : " + marks[3]);
          System.out.print("\n marks[4]  : " + marks[4]);
  }
}
```

Entering Data into an Array

- When more number of values are to be stored in an array, a for loop can be used.
- The sample code shows how to use a for loop in an array.

```
for(int i=0;i<5;i++)
{
        System.out.print("Enter the marks: ");
        str = stdin.readLine();
        marks[i] = Integer.parseInt(str);
}
```

Reading Data from an Array

- You can use a for loop with a single println statement to print the values from an array.

```
for (int i=0;i<5;i++)
{
        System.out.println("Marks : "+marks[i]);
}
```

Example:

```
import java.io.*;
class one_Int_array
{
   public static void main(String args[]) throws IOException
   {
          BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));
           String str;
           int marks[]=new int[6]; //array declaration

           //Accepting the marks
           for(int i=0;i<5;i++)
           {
              System.out.print("Enter mark"+(i+1)+" :");
              str = stdin.readLine();
              marks[i] = Integer.parseInt(str);
           }
           //Displaying the array
           for(int i=0;i<5;i++)
           {
               System.out.println("Marks "+(i+1)+" :"+marks[i]);
           }
   } }
```

Pop Quiz !!!

Identify the array index of each element.

1. *int ary[ ] = {2, 4, 6, 8}*
   a. 0, 1, 2, 3
   b. 1, 2, 3, 4
   c. 2, 4, 6, 8
   d. 0, 2, 4, 6

2. *int scores[ ] = new int[25];*

   Identify the valid elements in the following: Give reason for the same.

a. *scores[0]*

b. *scores[1]*

c. *scores[-1]*

d. *scores[25]*

Take

Activity

1. Write a Java program to find out the greatest and least values in an array.

Summary for 1- D Array

In this presentation, you learnt the following

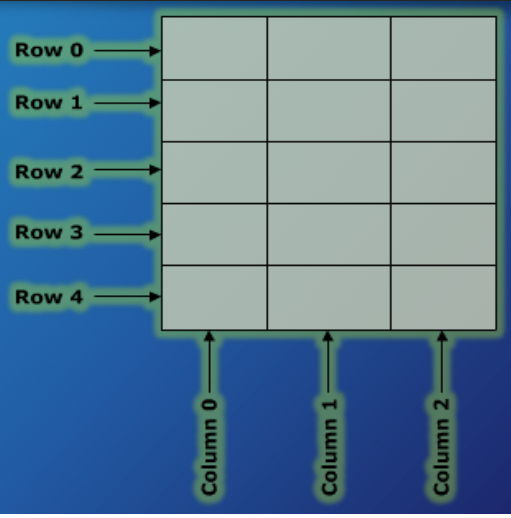
- Array is a collection of values of similar data types, stored in consecutive memory locations.
- Array index refers to the location of the values in an array.
- The first element in an array will have the array index as *0*.

- When an array is declared, the size of the array has to be mentioned.
- Arrays can be divided based on the number of subscripts used in the array. They are:
  - One-dimensional array.
  - Two-dimensional array.
- A one-dimensional array has one subscript.

TWO-DIMENSIONAL ARRAYS

- Are arrays with more than one dimension.
- Is a collection of a number of one-dimensional arrays placed one below the other.
- Two-dimensional arrays represent data in terms of rows and columns. It has two subscripts.

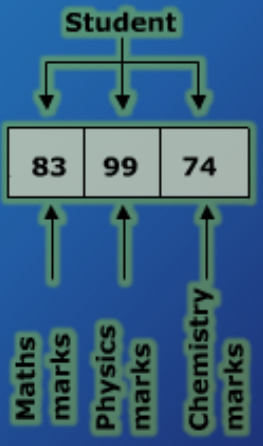
Representation – 2D Array



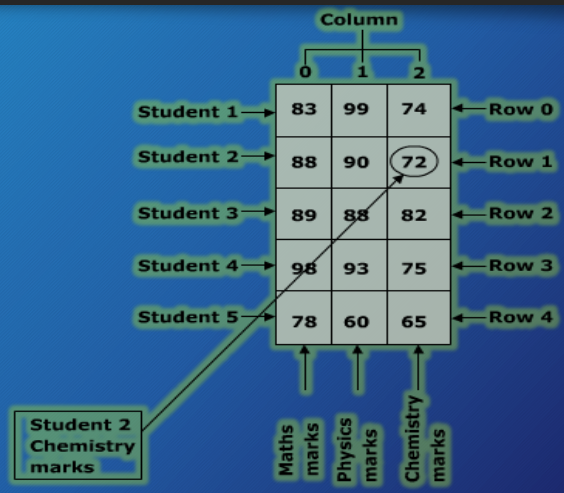
Declaring & Initialising 2D Array

- Assume that there are 5 students in a class and each of them study three different subjects, for example Mathematics, Physics and Chemistry.
- The marks of each student in all three subjects can be represented in a single row.

1-D Array Representation



2-D Array Representation



Declaration of 2-D Array

Example:

int marks_table[][] = new int[5][3];

Syntax:

<Data type> <Variable name>[][] = new <Data type> [Row][Column];

Accessing Array Elements

- You can use the row and column array index to access the array elements.
- For example, if you need to print the value stored in first row and second column of the array *marks_table*, (i.e. chemistry mark of student 2) the code will be :

Example:

System.out.println(marks_table[1][2]);

Example:

```
class twodim_array_students_marks
{
        public static void main(String args[])
        {
                int marks_table[ ][ ] = {
                                                {83,99,74},
                                                {88,90,72},
                                                {89,88,82},
                                                {98,93,75},
                                                {78,60,65}
                                        };
                System.out.println(marks_table[1][2]);
        }
}
```

Entering Data into Array

- A nested for loop can be used to enter data in a 2D array
- For example:

```
for (int x=0;x<2;x++) //row
{
        for(int y=0;y<3;y++) //column
        {
                m = stdin.readLine();
                marks_table[x][y] = Integer.parseInt(m);
        }
}
```

Reading Data from an Array

- Example to read data from an array

```
for(int x=0;x<2;x++)
{
        for(int y=0;y<3;y++)
        {
                System.out.println("Marks : "+marks[x][y]);
        }
}
```

Pop Quiz !!!

Activity:

Choose the correct declaration from the declaration statements given:

Declaration A	Declaration B	Declaration C	Declaration D
double table [ ][ ] = { 12, -9, 8, 7, 14, -32, -1, 0} ;	double table[ ][ ] = { {12, -9, 8}, {7, 14, 0}, -32, -1, 0} };	double table[ ][ ] = { {12, -9, 8} {7, 14} {-32, -1, 0} };	double table[ ][ ] = { {12, -9, 8}, {7, 14}, {-32, -1, 0} };

3. Which of the following statements constructs an array with 5 rows of 7 columns?

long stuff[ ][ ] = new stuff[5][7];

long stuff[ ][ ] = new long[5][7];

long stuff[ ][ ] = long[5][7];

long stuff[ ][ ] = long[7][5];

4. Which of the following statements that constructs a two-dimensional array with 7 rows?

int array[ ][ ] = new int[7][ ];

int array[ ][ ] = new int[][];

int array[ ][ ] = new int[ ][7];

int[] array[7] = new int[ ];

Array of Objects

Example :

```
class Student
{
   int marks;
}
```

- An array of objects is created just like an array of primitive type data items in the following way.

```
Student[] studentArray = new Student[7];
```

- The Student objects have to be instantiated using the constructor of the Student class and their references should be assigned to the array elements in the following way.

```
studentArray[0] = new Student();
```

- In this way, we create the other Student objects also. If each of the Student objects have to be created using a different constructor, we use a statement similar to the above several times. However, in this particular case, we may use a for loop since all Student objects are created with the same default constructor.

```
for ( int i=0; i<studentArray.length; i++)
{
    studentArray[i]=new Student();
}

studentArray[0].marks=100;
```

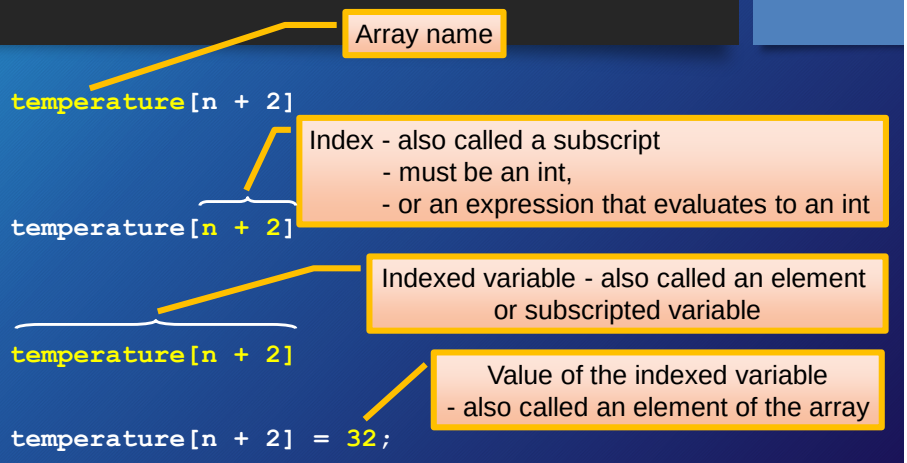
Example : Array of Objects

```
public class Student
{
   private String name;
   private int    id;
   private float  grade;

   public Student()
   { name = "none"; id = 0; grade = .0; }
   public Student(String name1, int id1, float grade1)
   { name = name1; id = id1; grade = grade1;}
}

public class Course
{
   private String name;
   private Student[] studentArray;

   public Course(String name1, int numOfStudents)
   {
      name = name1;
      studentArray = new Student[numOfStudents];
      for (int i = 0; i < numOfStudents; i++)
        studentArray[i] = new Student();//how to init name,id,grade for each obj
   }
}
```

Full code…

```
public static void main(String[] args)
{
   Student[] studentArray = new Student[7];
   studentArray[0] = new Student();
   studentArray[0].marks = 99;
   System.out.println(studentArray[0].marks); // prints 99
   modify(studentArray[0]);
   System.out.println(studentArray[0].marks); // prints 100 and not 99
}


public static void modify(Student s)
{
   s.marks = 100;
}
```

Some Array Terminology



Subscript Range

- Array subscripts use zero-numbering
  - the first element has subscript 0
  - the second element has subscript 1
  - etc. - the n^{th} element has subscript n-1
  - the last element has subscript `length-1`
- For example: an int array with 4 elements

Subscript:	0	1	2	3
Value:	97	86	92	71

Pass and return Array in methods

When Can a Method Change an Indexed Variable Argument?

- primitive types are "call-by-value"
  - only a copy of the value is passed as an argument
  - method *cannot* change the value of the indexed variable
- class types are reference types ("call by reference")
  - pass the address of the object
  - the corresponding parameter in the method definition becomes an alias of the object
  - the method has access to the actual object
  - so the method *can* change the value of the indexed variable if it is a class (and not a primitive) type

Passing Array Elements

```
int[] grade = new int[10];
obj.method(grade[i]);  // grade[i] cannot be changed

… method(int grade)     // pass by value; a copy
{
}
```

```
Person[] roster = new Person[10];
obj.method(roster[i]); // roster[i] can be changed

… method(Person p)      // pass by reference; an alias
{
}
```

Entire Arrays as Arguments

- Declaration of array parameter similar to how an array is declared
- Example:

```
public class SampleClass
{
    public static void incrementArrayBy2(double[] anArray)
    {
        for (int i = 0; i < anArray.length; i++)
            anArray[i] = anArray[i] + 2;
    }
    <The rest of the class definition goes here.>
}
```

Entire Arrays as Arguments

- Note – array parameter in a method heading does not specify the length
  - An array of any length can be passed to the method
  - Inside the method, elements of the array can be changed
- When you pass the entire array, do not use square brackets in the actual parameter

Example: An Array as an Argument in a Method Call

the method's argument is the name of an array of characters

```
public static void showArray(char[] a)
{
   int i;
   for(i = 0; i < a.length; i++)
      System.out.println(a[i]);
}

-------------
char[] grades = new char[45];
MyClass.showArray(grades);
```

uses the `length` attribute to control the loop
allows different size arrays and avoids index-out-of-bounds exceptions

Methods that Return Arrays

- A Java method may return an array
- Note definition of **return type as an array**
- To return the array value
  - Declare a local array
  - Use that identifier in the **`return`** statement

Methods that Return an Array

- the address of the array is passed
- The local array name within the method is just another name for the original array

```
public class returnArrayDemo
  {
      public static void main(String arg[])
      {
          char[] c;
          c = vowels();
          for(int i = 0; i < c.length; i++)
             System.out.println(c[i]);
       }
       public static char[] vowels()
       {
           char[] newArray = new char[5];
           newArray[0] = 'a';
           newArray[1] = 'e';
           newArray[2] = 'i';
           newArray[3] = 'o';
           newArray[4] = 'u';
           return newArray;
       }
  }
```

`c`, `newArray`, and the return type of `vowels` are all the same type: `char []`

Compile and Execute Java Program

-Practical Session

Next lecture…

- Design Class.
  - ➢Identify built-in classes in Java library.
  - ➢Define method in Java programs.
  - ➢Identify the use of methods in Java programs.
  - ➢Write methods in Java programs.


Don't depend too much on anyone in this world because even your own shadow leaves you when you are in darkness.
~ Ibn Taymiyyah
islamicway.tumblr
islaminpics.tumblr

PROGRAMMING LANGUAGE (JAVA)

Constructor

LESSON LEARNING OUTCOME:

At the end of this presentation, you will be able to:

- Use constructors to initialize data members

CONSTRUCTORS

- Is a **special method** that enables an object to initialise itself when it is created.
- It is used to **initialise the data members of a class**.
- Constructor gets **automatically called** when an object is created.

CHARACTERISTICS OF CONSTRUCTORS

- Constructors have the **SAME** name as the class name.
- They do **NOT** return any value.
- The syntax of a constructor is similar to that of a method.
- Once defined the constructor is automatically called immediately when the object is created before the *new* operator completes its job.
- Java creates a default constructor when you do not define a constructor.

EXAMPLE - CONSTRUCTOR

```
class Book
{
    String book_name;
    String author_name
    int no_of_pages;

    Book(String bname, String aname, int nopages)
    {
        book_name = bname;
        author_name = aname;
        no_of_pages = nopages;
    }
    public static void main(String args[])
    {
        Book b1 = new Book ("Java 2","Herbert Schildt",100);
     }
}
```

HANDS-ON!

- Program Book1.java illustrates the use of constructor in a class.

LAB EXERCISE

1. Create a class named Person. It should have instance variables to record name, age and salary. These should be of types String, int, and float. Use the *new* operator to create a person object.

   Create a constructor inside the class to initialize instance variables. In the main method pass values while creating object. Display the values using the displayData( ) method.

LAB EXERCISE

2. Create a class named *bank_account.* It should have instance variables to record the name of the depositor, account number, type of account and balance amount in the account. These should be of types String, float, String and float, respectively.

   Create a constructor inside the class to initialise instance variables. In the main method, while creating objects pass values. Display the values using *displayData( )* method. Display the name and balance using *displayData( )* method.

SUMMARY

In this session, you learnt the following:

- A constructor is a method that enables an object to initialize itself when it is created.

ASSIGNMENT

1. What is a constructor?
2. Rewrite the class calculator using a constructor

METHOD OVERLOADING

- Two or more methods having the same name but different signature.
- Methods which have same name but different parameters are called as overloaded methods.
- The parameters in overloaded methods should differ in at least one of the following:
  - Number of parameters
  - Data type of the parameters

HANDS-ON!

- Program m_o_load.java will illustrates method Overloading.

ACTIVITY (A)

```
class Over
{
      void display()
      {
            System.out.println("Melaka ");
      }

      void display(int i)
       {
             int j=0;
            while(j<i)
             {
                        System.out.println("Johor ");
                         j++;
            }
      }
       void display(String str,int i)
       {
              for (int j=1;j<=i;j++)
                 System.out.println(str);
      }
}
```

```
class Poly {
        public static void main(String args[])
        {
          Over obj=new Over();
          obj.display();
          obj.display(2);
          obj.display("Kuala lumpur ",3);
        }
}
```

ACTIVITY (B)

1. Fill in the blanks in the following program that uses overloaded methods:

```
// Program to multiply numbers using overloaded methods

class Product
{
   public ______ void main(String args[])
     {
            int result1;
            double result2;
            prod obj=new _____();
            result1=obj.multiply(12,13);
            result2=obj.multiply(2.5,3.2);
            _______.out.println(result1);
            System.out.println(result2);
     }
}
```

ACTIVITY (B) (CONTD…)

```
class prod
{
    ______ multiply(int No1, int No2)
     {
              return( No1 * No2 );
    }

     double ________(double No1, ________ No2)
    {
        return( No1 * No2 );
    }
}
```

CONSTRUCTOR OVERLOADING

- The concept of having two or more constructors with different signature in the same class is called *Constructor Overloading*.
- Two or more constructors in a class with same name and different parameters are called as overloaded constructors.

HANDS-ON!

- Program c_o_load.java illustrates constructor overloading.

HANDS-ON!

- Perform adv_mtdovrl.java to explain the advantage of method overloading.

HANDS-ON!

- Perform adv_consovrl.java to explain the advantage of constructor overloading.

ACTIVITY (A)

1. The following program displays the addition of 2 integers and 2 real numbers. Insert overloaded constructors in the blanks given in the following program:

ACTIVITY (A) (CONTD…)

```
class Numbers
{
    public static void main(String args[])
    {
        Sum obj1=new Sum(0,0);
        Sum obj2=new Sum(0.0,0.0);
        obj1.add(5,7);
        obj2.add(5.0,7.0);
    }
}
```

ACTIVITY (A) (CONTD…)

```
class Sum
{
    _____(int a,int b)
    {
        ________
        ________
    }

    ___(double c,double d)
    {
        _________
        _________
    }
```

```
    void add(int a,int b)
    {
        this.a=a;
        this.b=b;
        System.out.println
            ("The sum is " + (this.a + this.b));
    }

    void add(double c,double d)
    {
        this.c=c;
        this.d=d;
        System.out.println
            ("The sum is " + (this.c + this.d));
    }
}
```

LAB EXERCISE

6. Write a program using overloaded methods to print a multiplication table for an integer and a real number.
7. Write a program to print the absolute value of an integer and a real number using overloaded methods.

ASSIGNMENT

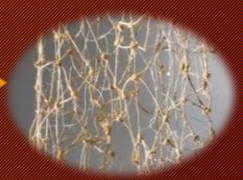
1. Define Constructor Overloading.
2. Write a program to find the Simple Interest. Use constructors to initialize the values of Principal, Number of years and rate of interest to 0.0.

You learn something every day if you pay attention.

~Ray LeBlond

VERYBESTQUOTES.COM

How Long Is A Piece Of String?



Object Oriented Programming (OOP)

Chapter 3:
Construct String in Java programs.

3

Lesson Learning Outcome

At the end of this class, you will be able to:

- Explain String references as parameters
- Differentiate between the **String** and **StringBuffer** class.
- Build Java programs using **String** and **StringBuffer** objects.
- Manipulate a **String** to a primitive data in Java programs.
- Construct program using method in **String** class.

String

Introduction to String

A string is a sequence of characters.

String Declaration

Example 1

```
char str [ ] = {'a','e','i','o','u'};
```

String Declaration

Example 2

```
char text[ ] = {'I',' ','C','A','N',' ','W','R','I','T',
'E','A',' ','G','O','O','D',' ','P','R','O','G','R','A','
M','I','N',' ','J','A','V','A'};
```

Example 3

```
String text = "I CAN WRITE A GOOD
PROGRAM IN JAVA";
```

String Class

- String class is present in java.lang package.
- The methods in this class are used to manipulate strings.
- The string objects created in a String class CANNOT be changed.

Creating String Object

Example 1

String s = new String( );

Syntax

String stringobject = new String( );

Creating String Object

Example 2

```
char arr[ ] = {'H','E','L','L','O'};
String s = new String(arr);
```

Syntax

```
String stringobject = new String(array_name);
```

Creating String Object

Example 3

```
char arr[ ] = {'H','E','L','L','O'};
String s = new String(arr);
String s1 = new String(s);
```

Syntax

```
String stringobject = new
String(stringobject);
```

Creating String Object

Example 4

String s = "I have to study for my exams";

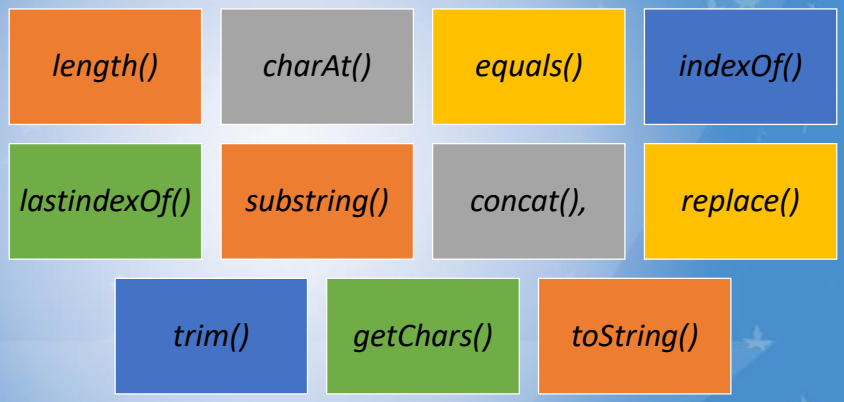
Syntax

String stringobject = <string value>;

String Methods

- The String class has several methods that can be used to manipulate the string values.
- Each method has its own characteristic.

Methods use in *String* class



length() method

- Is used to find the number of characters in a string.

length() method

Example 1

```
String s = "I am going to school";
int numchar = s.length( );
System.out.println(numchar);
```

Output

```
20
```

length() method

Example 2

```
String s = "  ";
int numchar = s.length( );
System.out.println(numchar);
```

Output

```
2
```

length() method

Syntax

int variable_name = stringobject.length( );

concat() method

Example 1

```
String firststring = "The Twin Tower";
String secondstring = " looks beautiful";
String thirdstring =
firststring.concat(secondstring);
```

Output

The Twin Tower looks beautiful

concat() method

Syntax

String Stringvariable3 = stringvariable1.concat (stringvariable2);

concat() method

Example 2

```
String s1 = "Result: "+5+1;
System.out.println(s1);
```

Output

Result: 51

concat() method

Example 3

```
String s2 = "Result: "+(5+1);
System.out.println(s2);
```

Output

```
Result: 6
```

charAt() method

- Extracts a single character from a string

charAt() method

Example

```
String s = "Malaysia"
char ch = s.charAt(4);
System.out.println(ch);
```

Output

```
y
```

charAt() method

Syntax

```
char variablename =
stringobject.charAt(int where);
```

equals() method

- Compares two strings for equality.

equals() method

Example

```
String s1="Both are equal";
String s2="Both are equal";
String s3="Both are not equal";
System.out.println(s1.equals(s2));
System.out.println(s1.equals(s3));
```

Output

```
true
false
```

equals() method

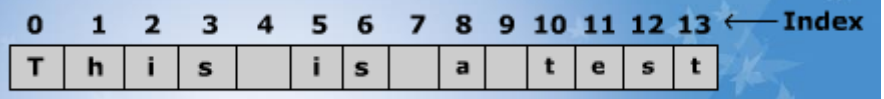
Syntax

boolean variablename = stringobject.equals(stringobject);

indexOf() method

- Will search the first occurrence of a character or set of characters inside a string.

indexOf() method



Example 1

```
int val1 = s.indexOf('s');
System.out.println(val1);
```

Output

3

indexOf() method

Example 2

```
int val2 = s.indexOf("is");
System.out.println(val2);
```

Output

2

indexOf() method

Syntax

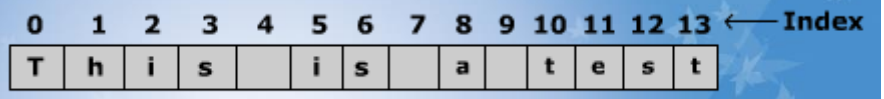
int variablename = stringobject.indexOf(character);

int variablename = stringobject.indexOf(string);

lastIndexOf() method

- Will search the last occurrence of a character or a set of characters inside a string.

lastIndexOf() method



Example 1

```
int val1 = s.lastIndexOf('s');
System.out.println(val1);
```

Output

12

lastIndexOf() method

Example 2

```
int val2 = s.lastIndexOf("is");
System.out.println(val1);
```

Output

5

lastIndexOf() method

Syntax

int variablename = stringobject.lastIndexOf(character);

int variablename = stringobject.lastIndexOf(string);

replace() method

- Will replace all occurrences of a character in a string with another character.

replace() method

Example

```
String s="Hewwo";
String s1 = s.replace('w','l');
System.out.println("Old string= " + s);
System.out.println("New string= " + s1);
```

Output

```
Hello
```

replace() method

Syntax

```
String stringvariable =
stringobject.replace(original,replacement);
```

trim() method

- Will trim the leading and trailing white spaces in a string.

trim() method

Example

```
String s="       Hello Malaysia    ";
String s1 = s.trim( );
System.out.println(s1);
```

Output

Hello Malaysia

trim() method

Syntax

String stringvariable = stringobject.trim( );

toLowerCase() method

- Converts all the characters in a string from uppercase to lowercase.

toLowerCase() method

Example

```
String s1="LOWER";
String s2 = s1.toLowerCase( );
System.out.println(s2);
```

Output

lower

toLowerCase() method

Syntax

```
String stringvariable =
stringobject.toLowerCase( );
```

toUpperCase() method

- Converts all the characters in a string from lowercase to uppercase.

toUpperCase() method

Example

```
String s1="upper";
String s2 = s1.toUpperCase( );
System.out.println(s2);
```

Output

UPPER

toUpperCase() method

Syntax

String stringvariable = stringobject.toUpperCase( );

substring() method

- Extracts a part of the string.

substring() method

Example

```
String s1="I do shopping in Mega Mall";
String s2 = s1.substring(5,12);
System.out.println(s2);
```

Output

shopping

Pop Quiz !!!

Question 1

1. Consider the string declared using the statement given:

   String line = "The sky is blue in color";

   Which of the following will convert the string to lowercase:

   a. *String a = line.toLowerCase();*
   b. *String b = toLowerCase(line);*

Question 2

2. Examine the following code:

   String s1 = "Wild";
   String s2 = "Flower";
   String s3 = "Rose";
   String Result;

Which of the following statements can be used to store the string Wild Flower Rose in Result?

A. *Result = s1.concat(s2.concat(s3));*
B. *Result.concat(s1,s2,s3);*

Summary of String

In this session, you learnt the following:

- A string is a sequence of characters.
- In Java, string is considered as a class.
- Objects of string class will be of string data type.
- Once you create string object you cannot change the characters that comprise the string.

Summary of String

In this session, you learnt the following:

- You can create a string instance using a string literal. For each string literal, Java automatically constructs a string object.
- The string class has several methods that can be used to manipulate the string values.
- There are methods to compare two strings, search for a substring, concatenate two strings and change the case of letters within a string.

StringBuffer

The *StringBuffer* Class

- Is a special class supported by Java to handle strings.
- It creates strings of flexible length whereas String class creates strings of fixed length.

StringBuffer vs *String* Class

1. once we creates a string object we can't perform any changes in the existing object. if we are trying to perform any changes with those changes a new object will be created. this non changeable nature is nothing but immutability of the string object.

2. Once we creates a StringBuffer object we can perform any type of changes in the existing object. this changeble is nothing but mutability of the StringBuffer object.

Creating *StringBuffer* Objects

Example 1

StringBuffer sb1 = new StringBuffer();

Syntax

StringBuffer StringBufferobject = new StringBuffer();

Creating *StringBuffer* Objects

Example 2

```
StringBuffer s = new StringBuffer(6);
```

Syntax

```
StringBuffer StringBufferobject = new
StringBuffer(integervariable);
```

Creating *StringBuffer* Objects

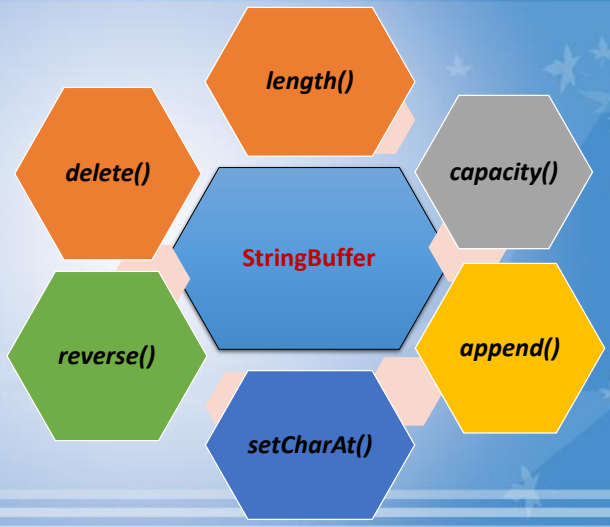
Example 3

```
StringBuffer sb3 = new StringBuffer ("Java");
```

Syntax

```
StringBuffer StringBufferobject = new
StringBuffer(stringvariable);
```

Methods use in *StringBuffer*



length() method

- Finds the current length of a StringBuffer object.

length() method

Example

```
StringBuffer sb1 = new System.out.println (sb1.length());
```

Output

12

capacity() method

- Returns the amount of memory allocated for the StringBuffer object.

capacity() method

Example

```
StringBuffer sb1 = new StringBuffer("Capacity of sb");
System.out.println (sb1.capacity());
```

Output

```
30
```

Hands-On!

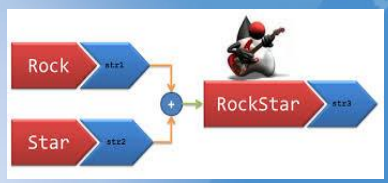
- Program *funcdemo1.java* will illustrate the use of *length()* and *capacity()* methods.

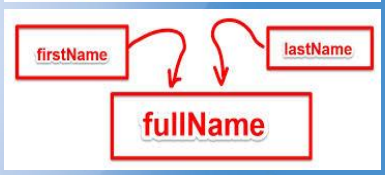
append() method

- Allows data to be added to the end of a StringBuffer object.

append()
vs
concat()

append() vs concat() functions





append() method

Example

```
StringBuffer sb1 = new StringBuffer("Welcome to beautiful");
sb1.append("_World");
System.out.println(sb1);
```

Output

Welcome to beautiful_World

Pop Quiz !!!

Question

Create a StringBuffer object and illustrate how to append characters. Display the capacity and length of the StringBuffer object.

setCharAt() method

- Sets the character at any position within a StringBuffer.

setCharAt() method

Example

```
StringBuffer sb1 = new StringBuffer("Hello World");
sb1.setCharAt(5,'_');
System.out.println(sb1);
```

Output

```
Hello_World
```

insert() method

- Allows data to be added at a specified position in the StringBuffer object.

insert() method

Example

```
StringBuffer sb1 = new StringBuffer("Hello World");
sb1.insert(6,"Java ");
System.out.println(sb1);
```

Output

Hello Java World

Hands-On!

- Program *funcdemo2.java* will illustrate the use of *append()* and *insert()* methods.

reverse() method

- Reverses the order of characters in a StringBuffer object.

reverse() method

Example

```
StringBuffer sb=new StringBuffer("roweT");
sb.reverse( );
```

Output

Tower

Hands-On!

- Program *ReverseItAll.java* will illustrate the use of *reverse()* method.

Pop Quiz !!!

Questions

1. Create a StringBuffer object and illustrate how to insert characters at its beginning.

2. Create a StringBuffer object and illustrate the operation of the *reverse()* method.

Summary

In this session, you learnt the following:

- StringBuffer will automatically grow to make room for additions and insertions.
- Characters can be inserted or appended to a StringBuffer object using its methods.

Questions

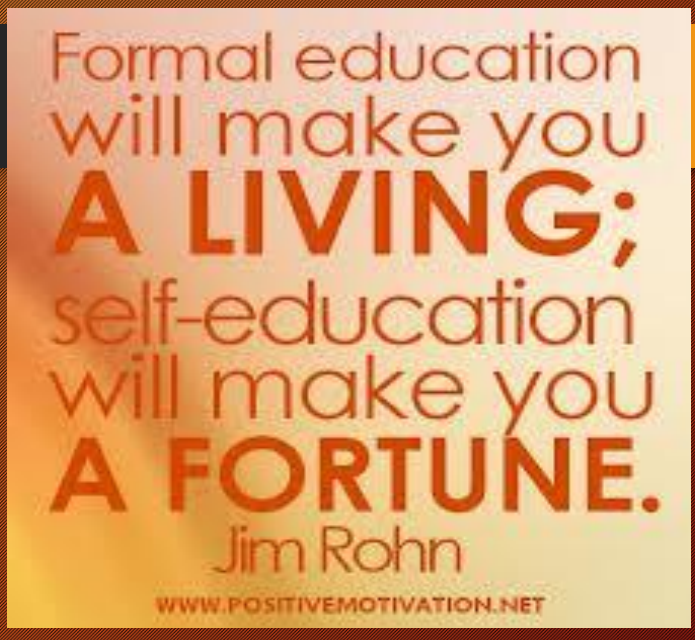
1. What is the difference between String object and StringBuffer object?
2. List any three methods in the StringBuffer class.

Lab activity for today….

-Practical Session (Lab Activity 5)

Next lecture…

- Chapter 3: Organize codes using Package.
- Chapter 4: Inheritance And Polymorphism



CHAPTER 3:

Organize Codes Using Package

Lesson Learning Outcome

At the end of this topic, you will be able to

- Describe packages and it uses.
- Explain built-in packages in Java programs.
- Explain naming conventions for packages.
- Build packages in Java programs

Reusability of Classes

- Java allows reusability of classes.
- Classes created for one program can be reused for another.
- It saves the development time.
- It is possible to use classes across packages even if the classes share the same name.
- You will be able to distinguish the classes with the help of the package name

Packages

- Is a **collection of classes** that can be shared by Java programs.
- Are classified into
  - *in-built packages*
  - *user-defined packages.*

Packages

- In-built packages - system packages.
- User-defined packages - Created by the users for their requirements.

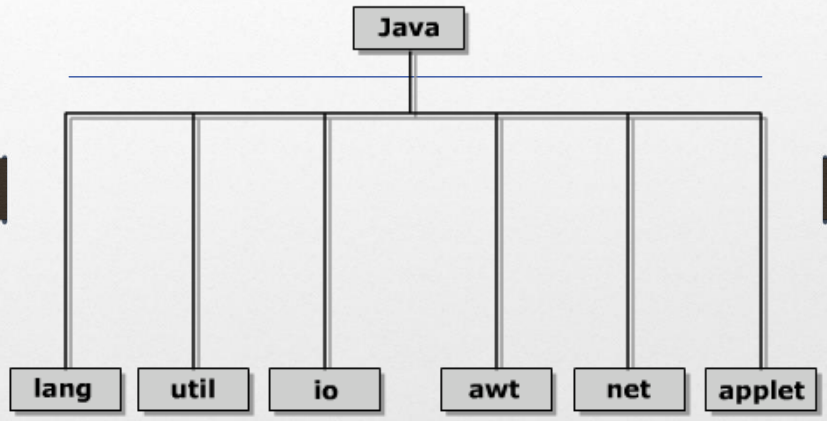
Advantages - Packages

- Allows reusability across programs.
- Allow classes to organize into units.
- Used to identify the classes.
- Same class name can be used in different packages.

Java Built-in Packages

- The classes are grouped together based on their functionality.
- Some of the important packages of Java are:
  - lang
  - util
  - io
  - awt
  - net
  - applet.

Important Java Packages



Important Java Packages - Description

Package Name	Description
lang	Provides language support classes.
util	Provides language utility classes such as vectors, hash tables random numbers and date.
io	Provides input and output support classes.

Cont…

Important Java Packages - Description

Package Name	Description
awt	Provides set of classes such as buttons, lists, menus, windows for implementing graphical interface
net	Provides classes for networking. Enables communication between the local computers and internet servers
applet	Provides classes for creating and implementing applets

java.lang package

- The *java.lang* package is one of the most important packages in Java.
- The *java.lang* package is automatically imported.
- It contains classes and interfaces that are fundamental to virtually all java programming.
- There are many classes in *java.lang* package.

Classes in java.lang Package

Class Name : *Math*

Description	Methods
It contains many mathematical functions	*static double sqrt(double a)* *static double pow(double a, double b)* *static int min(int a, int b)* *static int max(int a, int b)* *static double exp(double a)*

Classes in java.lang Package

Class Name : *String*

Description	Methods
It represents constants strings	*int length()* *char chatAt(int index)* *String concat(String str)* *String toLowerCase()* *String toUpperCase()*

Classes in java.lang Package

Class Name : *StringBuffer*

Description	Methods
It is used to represent a variable strings. It is useful when the string changes in value or in length.	*int length()* *int capacity()*

Classes in java.lang Package

Class Name : *Object*

Description	Methods
Inherits the variables and methods of object	*Object clone()* *Boolean equals(Object obj)*

Classes in java.lang Package (Wrapper classes)

Wrapper classes are applied for the following data types:

- *Boolean*
- *Character*
- *Integer*
- *Long*
- *Float*
- *Double*

Classes in java.lang Package

Class Name : *Boolean*

Description	Methods
Wraps the *boolean* fundamental data type.	*getBoolean()*

Classes in java.lang Package

Class Name : *Character*

Description	Methods
Wraps the *char* fundamental data type. It provides some useful methods for manipulating characters.	*static boolean isLowerCase(char ch)* *static boolean isUpperCase(char ch)* *static boolean isDigit(char ch)* *static boolean isSpace(char ch)* *static char toLowerCase(char ch)* *static char toUpperCase(char ch)*

Classes in java.lang Package

Class Name : *Integer*

Description	Methods
The *integer* class wraps the fundamental *integer* types int.	*static int parseInt(String s, int radix)* *static int parseInt(String s)* *static Integer getInteger(String name)*

Classes in java.lang Package

Class Name : *Long*

Description	Methods
It wraps the fundamental integer types *long*. It implements methods similar to those of the *Integer* class.	Similar to those of the Integer class.

Classes in java.lang Package

Class Name : *Float*

Description	Methods
The *Float* class wraps the fundamental floating-point types *float*.	*static int floatToIntBits(float value)* *static float intBitsToFloat(int bits)*

Classes in java.lang Package

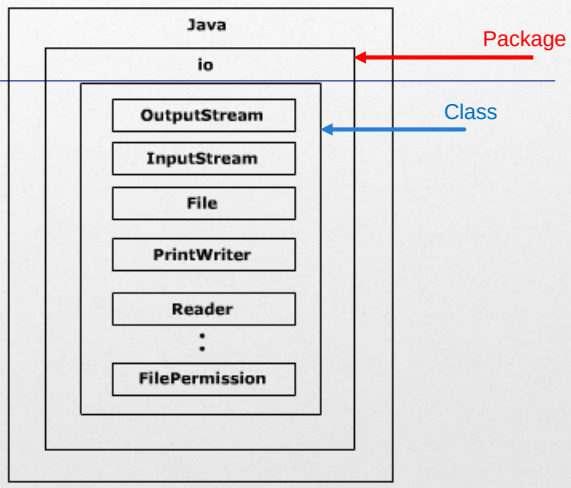
Class Name : *Double*

Description	Methods
The *Double* class wraps the fundamental floating-point types double.	*boolean isNaN()* *static boolean isNaN(float v)* *boolean isInfinite()*

Classes Within Packages

- A file name in java has the same name as the class in which main method is defined.
- All the files for the classes within a package have to be saved in a directory with the same name as that of the package.

Structure of io Package



The *import* Statement

- To access the classes from a package, the *import* statement is used.
- There are two ways to import packages.
  - Import all the classes of the package using *.
  - Import single class using the class name.

Importing all classes

- To import all the classes of a package * is used.

Example:

import java.io.;*

Syntax:

import java.packagename.;*

Importing specific class

- To import a specific class, the class name

Example:

import java.io.BufferedReader;

Syntax:

import java.packagename.classname;

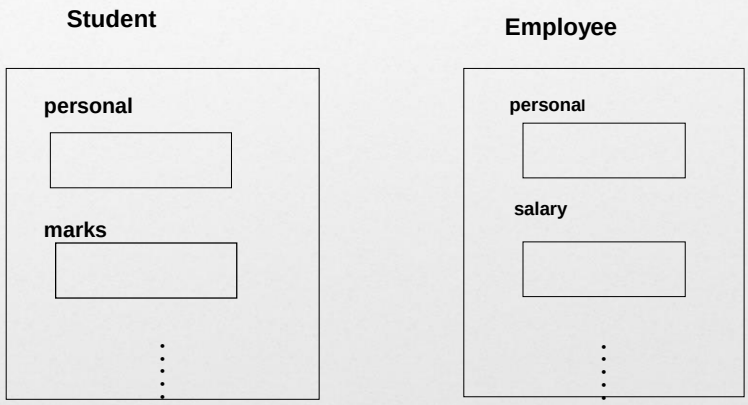
Hands-On!

- Program *InputSample.java* accepts input using input stream.

User-Defined Packages

- Users can create classes and group them into packages as per their requirement.
- In the in-built package, only the classes available in it can be used.
- In the case of user-defined packages, classes can be created and grouped into packages accordingly.

Package Structure



Naming Convention

- Selecting the package name depends upon how the classes are used.
- By naming convention, a package name begins with lowercase letter and a class name begins with uppercase letter.
- For example *java.lang.Math*, *lang* is the package name and *Math* is the class name.
- The naming convention helps to distinguish the package name and the class name easily.

Steps in creating packages

1. Declare the package in the first line of the source file. Package name must be in lowercase letter.
2. Declare the class as *public*, so that it can be used in other packages.
3. Define the methods of the class.
4. Create a subdirectory under the current directory.
5. Name the subdirectory same as the package name.
6. Store the source file in the subdirectory.
7. Compile the file. This file creates .class file in the subdirectory.

Package Declaration : Syntax

```
package packagename;
public class classname
{
    ……….
……….
……….
}
```

Package Declaration : Example

```
package  student;
public class Personal
{
 ……..
 ……..
 }
```

Importing User-Defined Packages

```
import student.*;
class Student_details
{
.....
.....
.....
}
```

Summary

In this class, you learnt the following:

- Package is a collection of classes that can be shared by Java programs.
- Packages provides reusability, they allow classes to be organized into units. They are used to identify the classes.
- Packages enable classes to have same name across them.

Summary

- The classes are grouped together based on their functionality.
- Some of the important java packages are *lang*, *util*, *io*, *awt*, net and *applet*.
- *import* statement is used to access the classes from the packages.

Summary

- The java.lang package is one of the most important packages.
- It provides number of classes and interfaces.
- Some of the important classes in the java.lang package are Boolean, Byte, Character, Class, Double, Float, Integer, Long, Math, Object, Short, String and StringBuffer.