CPT108 Data Structure and Algorithms

Lecture 1

Problem Analysis and Procedural Abstraction

Outline

Outline

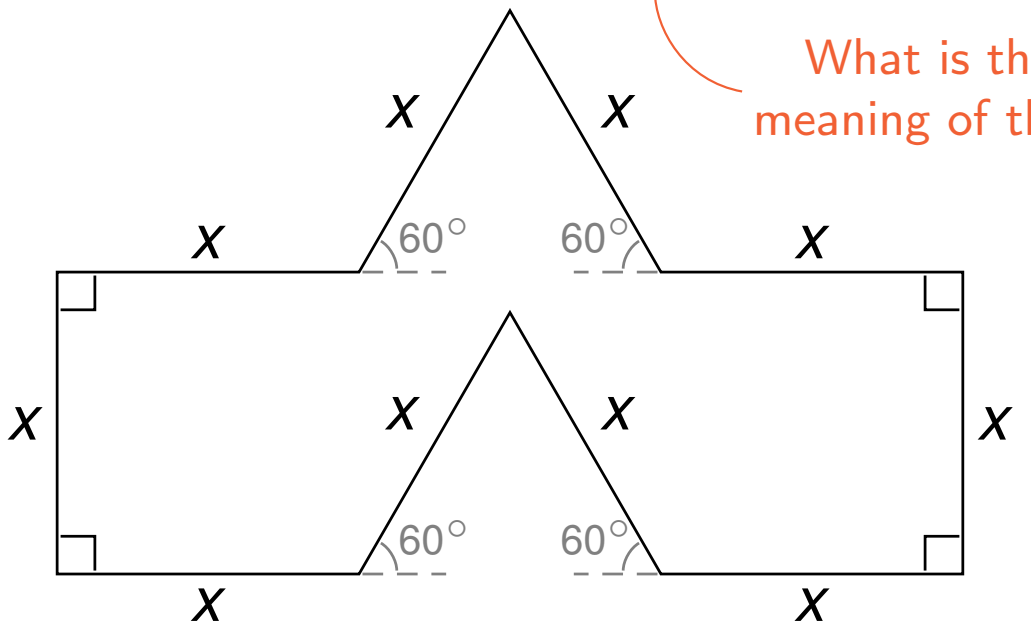
Problem Solving

- Problem Statement
  - What is the problem about?
- Problem Analysis
  - Input: What is *given*? What is *missing*?
    and How to *obtain* them?
  - Output: What is *required*?
  - Constraints: Under what *conditions*?
  - Abstraction: What information are *essential*?
- Solution Development
- Solution Validation / Verification

An Example

Divide the following shape into 5 equal pieces.

What is the meaning of this?

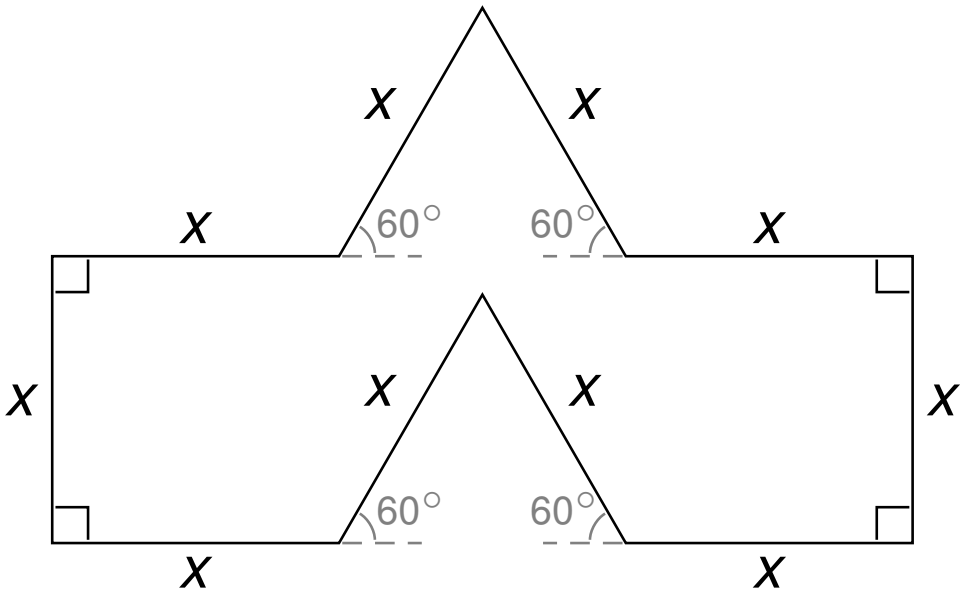


- Do you understand the problem above?
  - Is there any *ambiguity* in the question, or something that is *unclear* to you?

An Example (cont.)

What would you suggest to change in the question to remove the ambiguity and make it more clear?

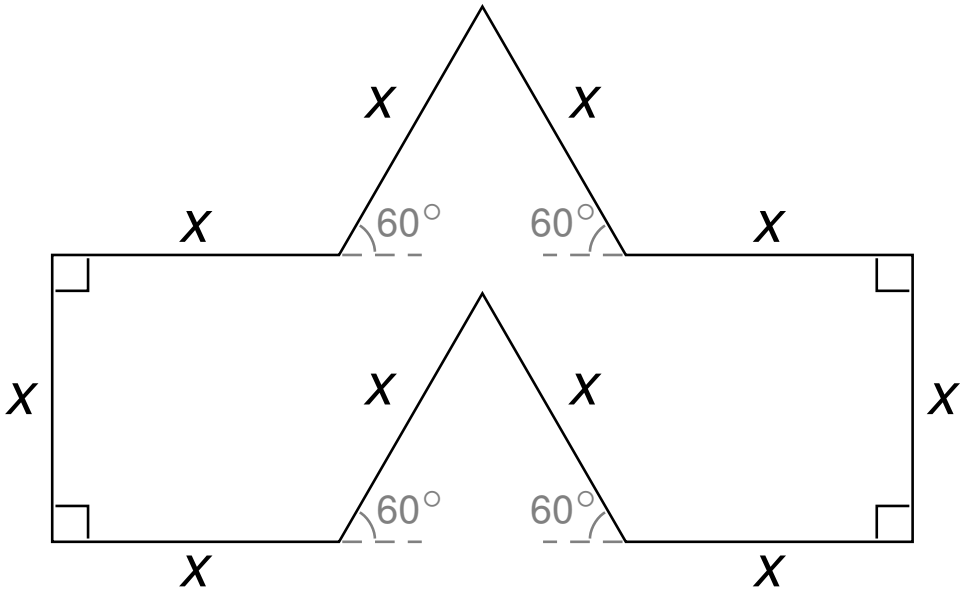
Divide the following shape into 5 ~~equal pieces~~ *identical shape*.



Any other ambiguities?

An Example (cont.)

Divide the following shape into 5 identical shape.



From the question:

- Can you identify the information that has been given by the problem statement, and
- What information can you *deduce* from the information that you have identifyed?

Programming as Problem Solving

The Program Development Process:

1. Problem Statement
   - What is the program trying to achieve?
2. Problem Analysis
   - Input: What is *given*? What is *missing*? and How to *obtain* them?
   - Output: What is *required*?
   - Constraints: Under what *conditions*?
   - Abstraction: What information are *essential*?
     - ⇒ *All must be presented in some form of data that can be **processed** by computer*
     - ⇒ Must be ***precise***
3. Program Design
   - Decomposition
   - Initial Algorithm Development
   - Algorithm Refinement

Our focus

4. Program Implementation (i.e., Coding)
   - Translate the algorithm into program using a programming language
5. Program Debugging & Testing

What is a good program?

- Correct
  - Meet the ***problem requirements***
  - Produce ***correct results***
- Easy to *read* and *understand*
- Easy to *modify*
- Easy to *debug* and *maintain*
- *Efficient*
  - Fast
  - Requires less memory

Typically referred to as computational and space complexities, respectively

GOOD Programming practiceing

Before you code / write the program, you should:

- Think and design carefully
- Check and prove the algorithm: Is it correct?
- Organize the program such that it can be easy to understand and debug

Check your program carefully in every step.

⇒ Reduce the number of *errors*

⇒ Reduce the amount of *time* spent in *debugging*

⇒ Produce ***GOOD*** programs

Bad Programming Practice

- Write code *without* *detailed analysis* and design
- Repeated *trial and error* *without* *understanding* the problem
  - Lost and frustrated after many failures
  - Even worst, cannot reuse written code but to rewrite the program from scratch
- Debug the program line by line, statement by statement
- Write and test code without checking
- Tempt to write tricky but dirty programs – difficult to understand, debug and modify

⇒ Spend enormous amount of time in ***debugging***

⇒ Produce ***POOR*** programs

An Example

Problem Statement

You are given a collection of nickels (US 5 cents) and pennies (US 1 cents) coins, find the number of Chinese yuan and 10-cent coins in exchange.

Problem Analysis

Input:

- nickels (US 5 cents) (integer) – count of nickels
- pennies (US 1 cents) (integer) – count of pennies

Output:

- dollars (integer) – count of Chinese yuan in return
- change (integer) – count of 10-cents coins

Constraints: *None*

An Example (cont.)

Program design

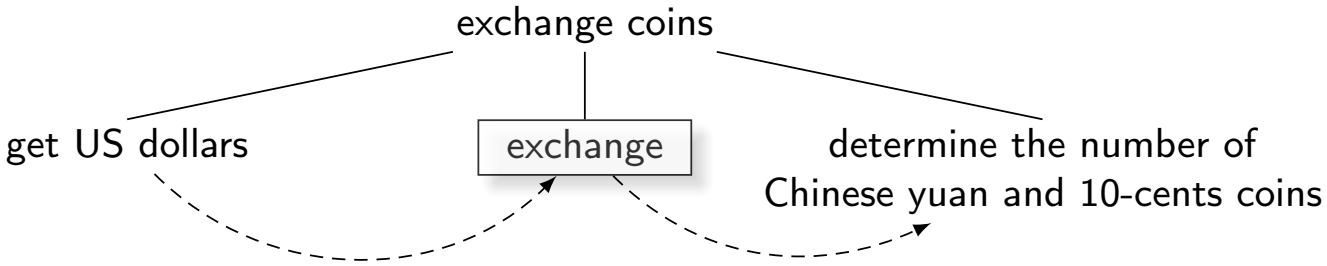
Knowledge:

- *one* dollar equals *100* cents

Decomposition:

- Find the total amount in US dollars
  - Input: count of nickels and pennies
  - Output: `total_US_dollars`
- Find the total amount of Chinese yuan in exchange
  - Input: `total_US_dollars`
  - Output: `total_CHN_dollars`
- Find the number of Chinese yuan and 10-cent coins
  - Input: `total_CHN_dollars`
  - Output: Chinese yuan and 10-cents coins change

Program Decomposition



New variables:

`total_US_dollars`

total amount
in US dollars

`total_CHN_dollars`

total amount in
Chinese yuan

Initial Algorithm:

1. Read in the count of nickels and pennies
2. Compute the total amount in US dollars
   - `total_US_dollars` = 5 $*$ nickels $+$ pennies
3. Compute the total amount in Chinese yuan in exchange
   - ??? How about this?
4. Find the value in Chinese yuan and change
   - `total_CHN_cents` (integer, total amount of 10-cents coins)
     - `total_CHN_cents` = `total_CHN_dollars` $*$ `DOLLAR2CENTS`
   - `yuan` = `total_CHN_cents` / `DOLLAR2CENTS`
   - `change` = `total_CHN_cents` % `DOLLAR2CENTS`
5. Display the value in Chinese yuan and change

Procedural Abstraction

To solve the exchange problem separately:

Function `exchange_us2chn()`

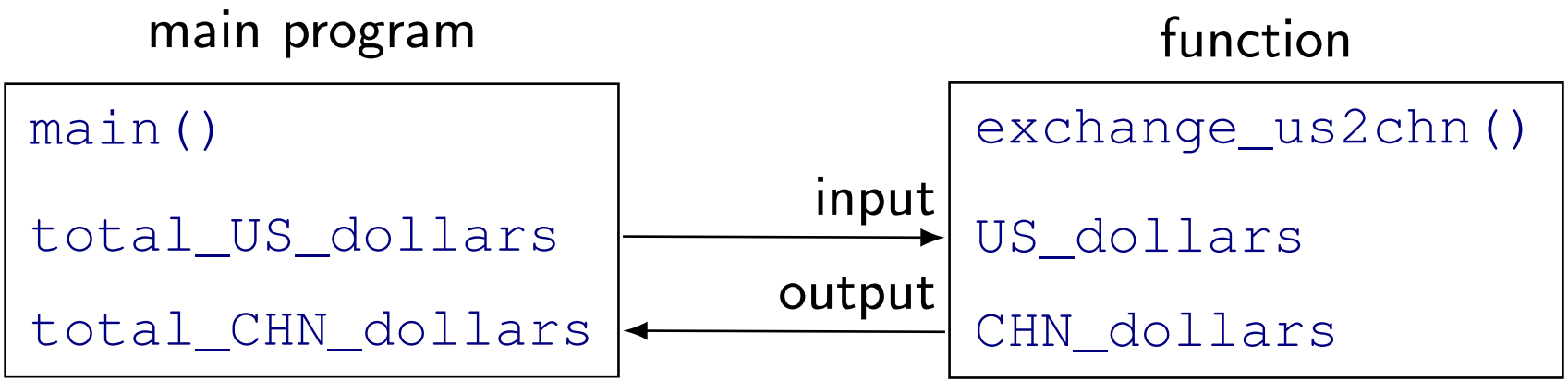
- Input: `US_dollars` (real) – amount of US dollars
- Output: `CHN_dollars` (real) – amount in Chinese yuan in exchange

At this point, we TEMPORARILY PUT ASIDE this subproblem.

Let us CONTINUE with our main problem solution.

Refinement 2 and Final Algorithm:

1. Read in the count of nickels and pennies
2. Compute the total amount in US dollars
   - `total_US_dollars` = 5 $*$ nickels $+$ pennies
3. Compute the total amount in Chinese yuan in exchange
   - `total_CHN_dollars = exchange_us2chn(total_US_dollars)` added the function to here!
4. Find the value in Chinese yuan and change
   - `total_CHN_cents = total_CHN_dollars * DOLLAR2CENTS`
   - `yuan = total_CHN_cents / DOLLAR2CENTS`
   - `change = total_CHN_cents % DOLLAR2CENTS`/10
5. Display the value in Chinese yuan and change



REMEMBER:

Before implementation, ***CHECK*** and ***DEBUG*** the algorithm!

Implementation: Part I

The Java main program skeleton:

```
package xjtlu.cpt108.basics;
public class MoneyExchange {
  // Compute the exchanged value from USD to Chinese yuan
  private static double exchange_us2chn(double US_dollars) {
    // TODO: implement the function
  }
  public static void main(String... arguments) {
    int nickels; // count of nickels
    int pennies; // count of pennies
    int yuan; // value of coins in Chinese yuan
    int change; // value of coins in Chinese 10-cents

    double total_US_dollars; // total US dollars
    double total_CHN_yuan; // total Chinese dollars

    // Read in the count of nickels and pennies

    // Compute the total amount in US dollars

    // Compute the total amount in Chinese dollars in exchange

    // Find the value in Chinese yuan and change

    // Display the value in Chinese yuan and change
  }
}
```

Implementation: Part I (cont.)

The final Java main program:

```
package xjtlu.cpt108.basics;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class MoneyExchange {

  private static final int DOLLARS2CENTS = 100;

  // Compute the equivalent Chinese yuan from USD
  private static double exchange_us2chn(double US_dollars) {
    // TODO: implement the function
  }

  public static void main(String... arguments) {
    int nickels; // count of nickels
    int pennies; // count of pennies
    int yuan; // value of coins in Chinese yuan
    int change; // value of coins in Chinese 10-cents

    double total_US_dollars; // total US dollars
    double total_CHN_yuan; // total Chinese dollars

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

How about this?

Implementation: Part I (cont.)

```
    try {
      // Read in the count of nickels and pennies
      System.out.println("Enter the number of nickels and press enter:");
      nickels = Integer.parseInt(br.readLine());
      System.out.println("Enter the number of pennies and press enter:");
      pennies = Integer.parseInt(br.readLine());

      // Compute the total amount in US dollars
      int total_US_cents = 5 * nickels + pennies;
      total_US_dollars = ((double) total_US_cents) / DOLLARS2CENTS;

      // Compute the total amount in Chinese dollars using the change rate
      total_CHN_yuan = exchange_us2chn(total_US_dollars);

      // Find the value in Chinese yuan and change
      int total_CHN_cents = (int) (total_CHN_yuan * DOLLARS2CENTS);
      yuan = total_CHN_cents / DOLLARS2CENTS;
      change = total_CHN_cents % DOLLARS2CENTS / 10;

      // Display the value in Chinese yuan and change
      System.out.printf("The change is %d Chinese yuan and %d 10-cents.\n",
        yuan, change);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Implementation: Part II

Solve the Subproblem

Function `exchange_us2chn()`

- Input: `US_dollars` (real) – amount of US dollars
- Output: `CHN_dollars` (real) – amount of Chinese yuan in exchange

Assumption:

- Known and fixed exchange rate
- `US2CHN` (real, constant) – US dollar to Chinese yuan exchange rate

Implementation: Part II (cont.)

Solve the Subproblem

Initial Algorithm:

1. Compute the equivalent Chinese yuan

Refinement and Final Algorithm:

1. Compute the equivalent Chinese yuan
   - `CHN_dollars = US_dollars * US2CHN`

```
private static final double US2CHN = 7.2;

// Compute the equivalent Chinese yuan from USD
private static double exchange_us2chn(double US_dollars) {
  return US2CHN * US_dollars;
}
```

Outline

Program Debugging and Testing

- Compile and run the program, test on different and random input data.
- Sources of errors:
  - Program can't be compiled – *syntax error*
    - Careless in coding, relatively easy to fix
  - Program can't be linked – *link error*
    - Forget or use wrong language libraries
  - Program is incorrect – *run time error*
    - Implementation (coding) error (e.g., wrong output, division by zero) $\rightarrow$ Locate the bugs and fix
  - *Algorithmic error*
    - Should not happen if check and prove carefully *before* implementation

Program Debugging and Testing (cont.)

- Always think and debug program **TOP-DOWN**
- Pay special attention to INTERFACES between the major steps/functions
  1. input → `nickels`, `pennies`
  2. `nickels`, `pennies` → `total_US_dollars`
  3. `total_US_dollars` → `total_CHN_dollars`
  4. `total_CHN_dollars` → `dollars`, `change`
  5. `dollars`, `change` → output
- Generate test cases:

```
Enter the number of nickels and press return:  21
Enter the number of pennies and press return:  105
The change is 18 Chinese yuan and 7 10-cents.

Enter the number of nickels and press return:  3
Enter the number of pennies and press return:  2
The change is 1 Chinese yuan and 2 10-cents.
```

Alternate Program Design

Final Algorithm:

1. Read in the count of nickels and pennies
2. Compute the total amount in US dollars
   - `total_US_dollars` = 5 $*$ nickels $+$ pennies
3. Compute the total amount in Chinese yuan in exchange
   - ~~`total_CHN_dollars = exchange_us2chn(total_US_dollars)`~~
   - `total_CHN_dollars = total_US_dollars * US2CHN`

   Compute the value directly here
4. Find the value in Chinese yuan and change
   - `total_CHN_cents = total_CHN_dollars * DOLLAR2CENTS`
   - `yuan = total_CHN_cents / DOLLAR2CENTS`
   - `change = total_CHN_cents % DOLLAR2CENTS`/10
5. Display the value in Chinese yuan and change

Alternate Program Design (cont.)

```
package xjtlu.cpt108.basics;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class MoneyExchange_refine {

  private static final int DOLLARS2CENTS = 100;

  private static final double US2CHN = 7.2;

  // Compute the equivalent Chinese yuan from USD
  private static double exchange_us2chn(double US_dollars) {
    return US2CHN * US_dollars;
  }

  public static void main(String... arguments) {
    int nickels; // count of nickels
    int pennies; // count of pennies
    int yuan; // value of coins in Chinese yuan
    int change; // value of coins in Chinese 10-cents

    double total_US_dollars; // total US dollars
    double total_CHN_yuan; // total Chinese dollars

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

function removed!

Alternate Program Design (cont.)

```
    try {
      // Read in the count of nickels and pennies
      System.out.println("Enter the number of nickels and press enter:");
      nickels = Integer.parseInt(br.readLine());
      System.out.println("Enter the number of pennies and press enter:");
      pennies = Integer.parseInt(br.readLine());

      // Compute the total amount in US dollars
      int total_US_cents = 5 * nickels + pennies;
      total_US_dollars = ((double) total_US_cents) / DOLLARS2CENTS;

      // Compute the total amount in Chinese dollars using the change rate
      total_CHN_yuan = total_US_dollars * US2CHN;

      // Find the value in Chinese yuan and change
      int total_CHN_cents = (int) (total_CHN_yuan * DOLLARS2CENTS);
      yuan = total_CHN_cents / DOLLARS2CENTS;
      change = total_CHN_cents % DOLLARS2CENTS / 10;

      // Display the value in Chinese yuan adn change
      System.out.printf("The change is %d Chinese yuan and %d 10-cents.\n", yuan
            , change);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

code updated

Procedural Abstraction – Why?

1. Allows TOP-DOWN DESIGN
   - Allows decomposition into *smaller* subproblems
   - Subproblems are *simpler* than the original problem
   - Allows RECURSION
     - When the subproblem is just the same but smaller problem
     - A very powerful problem solving technique
   - Subproblems can be solved *separately*, at *different time*, by *different person*

Procedural Abstraction – Why? (cont.)

2. For complex subproblems
   - *Information hiding* – hiding unwanted details from the main problem
   - Easy to *understand*
   - Easy to *modify*
     - The whole module/function can be replaced without affecting the main program code
3. For frequency encountered subproblems
   - Allows code *reuse*
   - Only *one* place to modify

Outline

Data Type

- An elementary data abstraction
- Captures the very nature of the data being considered

Simple data types (in Java)

Data type	Size	Description
`byte`	1 byte	Stores whole numbers from -128 to 127
`short`	2 bytes	Stores whole numbers from -32,768 to 32,767
`int`	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
`long`	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`float`	4 bytes	Stores floating point numbers. Sufficient for storing 6 to 7 digits
`double`	8 bytes	Stores floating point numbers. Sufficient for storing 15 decimal digits
`boolean`	1 bit	Stores `true` or `false` values
`char`	2 bytes	Stores character/letter or ASCII values

Data Type (cont.)

A special type

- `void` – no type (nothing)

Type consistency

- variable type and value must be consistent

Arithmetic rule

operand types	result type
`int, int`	`int`
`real, int`	`real`
`real, real`	`real`

Type Conversion (casting)

```
(type) variable
```

```
total_US_dollars = ((double) total_US_cents) / DOLLARS2CENTS;
(int) (total_CHN_yuan * DOLLARS2CENTS);
```

Data Type (cont.)

Constant Declaration

```
static final type variablelist
```

```
static final int DOLLARS2CENTS = 100;
static final double US2CHN = 7.2;
```

Variable Declaration

```
type variablelist
```

```
int nickels, pennies;
double total_US_dollars;
char char1;
String str;
```

Scope of variables

In most modern programming language, variables are only accessible inside the region, a.k.a. *scope*, they are created.

```
package xjtlu.cpt108.basics;
public class VariableScope {
  public static enum MessageType {
    SIMPLE_MESSAGE, ERROR_MESSAGE
  };

  private int i = 1;

  public void foo() {
    int i = 2;
    System.out.println(i);
    {
      double d = Double.valueOf(1.234);
      System.out.println(d);
    }
    System.out.println(d);
  }
}
```

global (publicly accessible)

global (within the class)

local variable (within the function)

what is the value of `i` here?

local variable (dynamic heap object)

no problem as `d` is valid here!

Error as `d` is no longer valid here!

- Accessibility of Variables and Functions can be changed according to the modifiers used.
- In Java, there are four different access levels in which the entity is defined and the package that contains the entity.

Modifier	Package	Subclass	Public
`public`	Yes	Yes	Yes
`protected`	Yes	Yes	No
no modifier	Yes	No	No
`private`	No	No	No

Note that the accessibility of variables and methods based on the modifiers varies with respect to the programming languages.

As a rule of thumb, you should check the accessibility rule of the programming language before using it!