A1

1. Classification of programming languages

1.1 Imperative (ภาษาเชิงคำสั่ง) → ทำให้คอมพิวเตอร์ประมวลผลอย่าง "เป็นลำดับขั้นตอน"

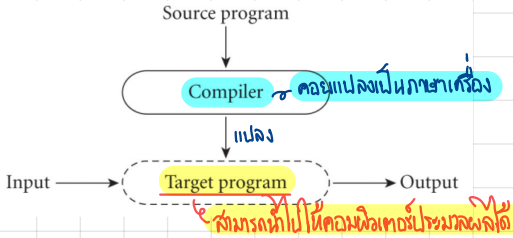
ลำดับการประมวลผลมีผลต่อผลลัพธ์ของการทำงาน

- Von Neumann : จัดให้โปรแกรมเป็นข้อมูลประเภทหนึ่ง เช่น Fortran, Pascal, C
- Scripting : เขียนสั้น+ง่าย เกิดจากการเรียงคำสั่งที่มีมาก่อนอยู่แล้ว สามารถนำไปประมวลผลกับสภาพแวดล้อมที่เฉพาะในลักษณะของ Interpreter เช่น Python, Ruby, JavaScript, PHP (รวมถึง dynamic general-purpose language)
- Object-oriented : ส่วนหลักจะว่าด้วย Class หรือ Object ซึ่งบรรจุ method ไว้ภายใน
  - ความสัมพันธ์ระหว่าง class เป็นแบบ semi-independent (อิสระ แต่ก็มีความสัมพันธ์กันอยู่) เช่น C++, JAVA

1.2 Declarative (ภาษาเชิงประกาศ) → ไม่ได้กำหนดลำดับขั้นตอน แต่จะกำหนดคุณสมบัติของคำตอบของปัญหา

- Functional : Based on recursive definition of functions
  Program is a function from inputs to outputs เช่น Lisp, Scheme, ML, Haskell
- Logic : บอกคุณสมบัติการแก้ปัญหาในรูปแบบของ rule เช่น Prolog

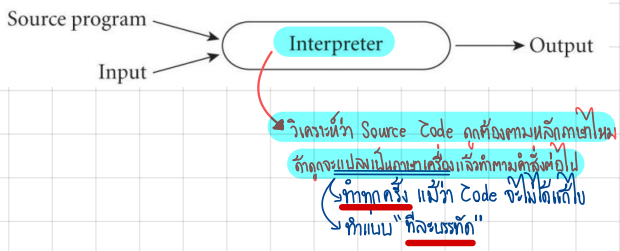
2. Program Translation : Compiler



ข้อดี : โปรแกรมเร็ว (เพราะการตัดสินใจบางอย่างที่ได้ตั้งแต่ขั้น Compile เช่น การกำหนด memory address ของตัวแปร) ฝังไว้ใน target program ได้เลย ทำให้ตอนทำงานระบุตำแหน่งได้ทันที

ข้อเสีย : ไม่ Flexible (ต่างเครื่อง ก็ต่างกัน)

3. Program Translation : Interpreter

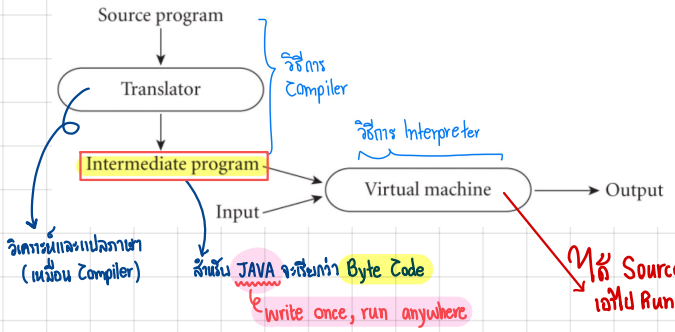


ข้อดี : แก้ไขข้อผิดพลาดง่าย + Flexible

ข้อเสีย : ช้า (ถ้าเป็น loop จะช้ามาก)

หมายเหตุ! ไม่ได้แปลเป็นภาษาเครื่องโดยตรง แต่จะเตรียม function ต่างๆ ในรูปของภาษาเครื่องไว้ก่อนแล้ว และจะเรียกฟังก์ชันเหล่านี้เพื่อแปลงภาษาแทน

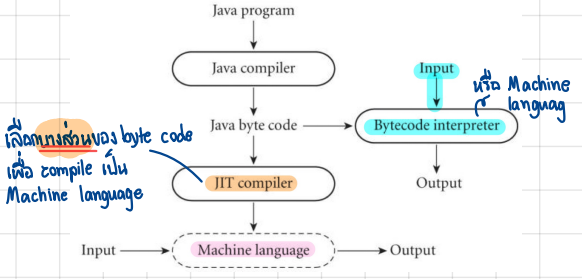
4. Program Translation : Compiler and Interpreter



ข้อดี : เคลื่อนย้ายได้ง่าย (สามารถนำไปทำบน platform ใดก็ตามที่มี Virtual Machine ติดตั้งอยู่) ไม่ต้อง compile ใหม่/ไม่ต้องเขียนโปรแกรมเพิ่ม

ข้อเสีย : ต้องดำเนินการแปลในทุกข้อความสั่ง (ช้า)

5. Just-in-Time Translation



- ทำให้ทำได้เร็วขึ้นโดยที่ไม่ต้องทำในขั้นตอนแปล
- คำสั่งไหนถูกเรียกใช้บ่อยจะถูก Compile เป็น Machine Language แล้วเก็บไว้ในแคช
- คำสั่งที่ไม่ถูกเรียกบ่อยจะถูกกระทำผ่าน interpreter
- มักพบใน modern language

A2

Binding = การยึดเหนี่ยวชื่อกี่ตั้งขึ้นกับสิ่งที่อ้างถึง

Binding time = time at which an association is created

decision is made) ← ภาษาส่วนใหญ่จะเป็นแบบนี้ เช่น C, C++, JAVA

Static binding = Things are bound before run time (early binding) → ข้อดี: ไม่ flexible เพราะเปลี่ยน data type

Associated with greater efficiency, e.g. compiler decides on layout of variables in memory and generates efficient code to access them.

Compiled languages tend to have early binding times. ← ติดตอน Run เลย ว่าจะจองพื้นไหนตรงไหน

Dynamic binding = Things are bound at run time (late binding)

Associated with greater flexibility, e.g. decision on which data value of which type is bound to a variable

name may be made at run time ← ข้อดี: เวลาเปลี่ยนประเภท data type

Interpreted languages tend to have later binding times. → ข้อดี!

↳ ภาษาที่ไม่มี Complier

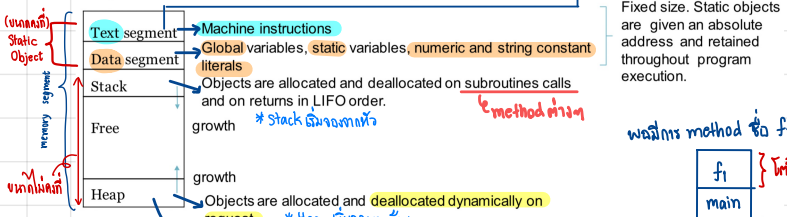
Object Lifetime and Storage Managment

Lifetime is time between creation and destruction.

An object to which a name is bound has its object lifetime.

Object lifetime corresponds to its storage allocation.

เหมือนจะพูดแค่เป็น โค้ดกี่เครื่องเข้าใจ ก็เก็น code ที่แปลงเป็น machine code (Source code)



↳ method ต่างๆ

ขณะมีการ method ชื่อ f1

โดน ↑ ทำให้

ส่วนของ memory ที่ใช้เก็บ data ที่เราต้องมีการ "สั่งจอง" ด้วยคำสั่งเฉพาะ เช่น new ใน JAVA (ส่วนมากเป็น Pointer)

เนื่องจากข้อนี้โจทย์สมมติว่า switch can use range นั่นคือเราสามารถฟันน case เป็นช่วงได้ ดังนั้นข้อนี้ควรใช้ switch เพราะ switch จะมีการสร้าง jumb table ทำให้เมื่อเราป้อนข้อมูลเข้าไป จะกระโดดไปที่คำตอบทันที (ไม่ต้องเช็กทีละครั้งแบบ if-else)

STATIC VS DYNAMIC METHOD (JAVA)

S.N.	Static Binding	Dynamic Binding
1.	A type of polymorphism that collects the information to call a method during the compile-time.	A type of polymorphism that collects the information to call a method at the runtime.
2.	The binding happens at compile time.	The binding happens at runtime.
3.	The actual object is not used for binding.	The actual object is used for binding. → Object จริง! ที่ new มา
4.	It is also called early binding because binding happens during compilation.	It is also called late binding because binding happens at run time.
5.	The execution speed is high.	The execution speed is slow.
6.	Method overloading is the best example of static binding.	Method overriding is the best example of dynamic binding.
7.	The methods which are private, static and final, show static binding because we can not override them.	The methods other private, static and final methods show dynamic binding because overriding is possible with these methods. → ถ้าเป็น C++ คือเป็น virtual

Code to understand the static binding in Java

```
1.  package com.techvidvan.binding;
2.  class Person
3.  {
4.    public void speak()
5.    {
6.      System.out.println("Person speaks");
7.    }
8.  }
9.  class Teacher extends Person
10. {
11.   public static void speak()   → override ไม่ได้ ∴ static method
12.   {
13.     System.out.println("Teacher speaks");
14.   }
15. }
16. public class StaticBinding
17. {
18.   public static void main( String args[ ])
19.   {
20.     // Reference is of Person type and object is Teacher type
21.     Person obj = new Teacher();   → object
22.     obj.speak();
23.     //Reference and object both are of Person type.
24.     Person obj2 = new Person();
25.     obj2.speak();
26.   }
27. }
```

ใช้ method ที่มี method เฉพาะ teacher ไม่ได้

Output:

```
Person speaks
Person speaks
```

Code to understand the Dynamic Binding in Java:

```
1.  package com.techvidvan.binding
2.  class Person
3.  {
4.    public void speak()
5.    {
6.      System.out.println("Person speaks");
7.    }
8.  }
9.  class Teacher extends Person
10. {
11.   @Override
12.   public void speak()
13.   {
14.     System.out.println("Teacher speaks");
15.   }
16. }
17. public class DynamicBinding
18. {
19.   public static void main( String args[])
20.   {
21.     //Reference and objects are of Person type.
22.     Person obj2 = new Person();
23.     obj2.speak();
24.     // Reference is of Person type and object is Teacher type
25.     Person obj = new Teacher();
26.     obj.speak();
27.   }
28. }
```

Output:

```
Person speaks
Teacher speaks
```

Type Clash ?

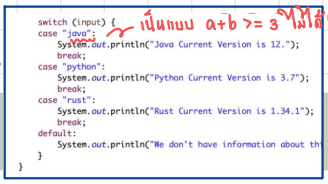
```
System.out.println(2 + 3 + ">=" + 1 + 1);
```

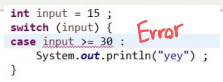
ไม่ใช่ ~~5 >= 2 นะ~~

expressions are evaluated from left to right, in this case 2 + 3 get summed to 5 and when "added" to a string result in `"5 >="`, which when added to 1 gives `"5 >= 1"`, add another 1 and your result is: `"5 >= 11"`

If-else Vs Switch

	If-else	switch
Definition	Depending on the condition in the 'if' statement, 'if' and 'else' blocks are executed.	The user will decide which statement is to be executed.
Expression	It contains either logical or equality expression.	It contains a single expression which can be either a character or integer variable. (เป็น logical ไม่ได้)
Evaluation	It evaluates all types of data, such as integer, floating-point, character or Boolean.	It evaluates either an integer, or character.
Sequence of execution	First, the condition is checked. If the condition is true then 'if' block is executed otherwise 'else' block	It executes one case after another till the break keyword is not found, or the default statement is executed.
Default execution	If the condition is not true, then by default, else block will be executed.	If the value does not match with any case, then by default, default statement is executed.
Editing	Editing is not easy in the 'if-else' statement.	Cases in a switch statement are easy to maintain and modify. Therefore, we can say that the removal or editing of any case will not interrupt the execution of other cases.
Speed	If there are multiple choices implemented through 'if-else', then the speed of the execution will be slow.	If we have multiple choices then the switch statement is the best option as the speed of the execution will be much higher than 'if-else'.





นั่นเอง ซึ่งจะเห็นว่าจะใช้ค่าที่สนใจเพื่อเลือกที่อยู่ที่จะกระโดดไปทำงาน ณ บรรทัดที่ต้องการได้เลย โดยไม่ต้องคอยตรวจสอบทีละ case ว่าค่าที่สนใจตรงกับเงื่อนไขหรือเปล่าทำให้สามารถทำงานได้เร็วกว่าเมื่อเทียบกับ nested if

```
int input = 15 ;
switch (input) {
case 15 :
    System.out.println("yey") ;
case 16:
    System.out.println("yey2") ;
case 17:
    System.out.println("yey3") ;
}
```

ถ้าไม่มี break จะแสดงผลลัพธ์ออกมาหมดเลย (Fall through)