មជ្ឈមណ្ឌលកូរ៉ េសហ្វវែរ អេច អ ឌី
Korea Software HRD Center

មូលនិធិ សហ្វវែរ កូរ៉ េសម្រាប់ជំនួយជាសកល
Foundation for Korea Software Global Aid

ប្រធានបទ៖

JAVA 8 - 11

អ្នកណែនាំ៖ លោក កឿន ដូច
លោក ឡេង ហុងម៉ែង
លោក ស៊ាប ឡងឌី

ប្រធានបទ៖

JAVA 8 - 11

សមាជិក៖

១. សារិនសុខ ម៉ានីន	៤. សួន ដារីតា
២. កៅ សុខមាន	៥. ឆាយ គីនីន
៣. អ៊ុន បូណាលីហេង	៦. លីម វាយោ

ថ្នាក់ សៀមរាប

Group 3

មាតិកា

១. ស្វែងយល់ពី Java 8 - 11

    ១.១ ស្វែងយល់ពី Functional Programming vs OOP

    ១.២ ស្វែងយល់ពី Learn new technologies of Java

២. ស្វែងយល់ពី Lambda Expression:

    ២.១ ស្វែងយល់ពី Working with Lambda

៣. ស្វែងយល់ពី Method References

មាតិកា

៤. ស្វែងយល់ពី Functional Interface

៥. ស្វែងយល់ពី Default method

៦. ស្វែងយល់ពី New Date Time API

៧. ស្វែងយល់ពី Base64

    ៧.១ ស្វែងយល់ពី How to encrypt content with Base64

៨. ស្វែងយល់ពី Stream API

៩. ស្វែងយល់ពី forEach() Method

មាតិកា

១០. ស្វែងយល់ពី Optional

១០.១ What is Optional

១០.២ Why use Optional instead of try-catch, if-else

១០.៣ How to use Optional

១០.៤ Understand using Optional

១០.៥ Optional Methods

១. ស្វែងយល់ពី Java 8 - 11

- **Function Programming Java 8 (Release in March 2014)** មានដូចជា:
  - **Lambda Expressions**
  - **Stream API**
  - **Functional Interface**

១. ស្វែងយល់ពី Java 8 - 11 (ត)

- Java Platform Modules System (JPMS) Java 9 (Release in September 2017) មានដូចជា:
  - Modular System - Jigsaw Project
  - A New HTTP Client
  - Process API
  - Small Language Modifications
  - Jshell Command Line Tool

១. ស្វែងយល់ពី Java 8 - 11 (ត)

- Java Platform Modules System (JPMS) Java 9: (ព)
  - JCMD Sub-Commands
  - Multi-Resolution Image API
  - Variable Handles
  - Publish-Subscribe Framework
  - Unified JVM Logging
  - New API

១. ស្វែងយល់ពី Java 8 - 11 (ត)

Feature in Java 10

- Java 10 (Release in March 2018) មានដូចជា:
  - Time-Based Release Versioning (JEP 322)
  - Local Variable Type Inference (JEP 286)
  - Experimental Java Based JIT Compiler (JEP 317)
  - Root Certificates (JEP 319)
  - Heap Allocation on Alternative Memory Devices (JEP 316)

១. ស្វែងយល់ពី Java 8 - 11 (ត)

Feature in Java 10

- **Java 10** មានដូចជា (ត):
  - **Deprecation and Removals:** Command line options and tools, និង API
  - **Unmodifiable Collections:** copyOf(Collection) method and toUnmodifiable() method
  - **Optional API orElseThrow() method**

១. ស្វែងយល់ពី Java 8 - 11 (ត)

Feature in Java 11

- **Feature នៅក្នុង Java 11 (Release in September 2018)** មានដូចជា:
  - **Oracle vs. Open JDK**
  - **Developer Feature មានដូចជា:**
    - **New String method:** Java 11 បន្ថែម method ទៅកាន់ String class ដូចជា: isBlank(), lines(), strip, stripLeading, stripTrailing, and repeat ។
    - **New file Methods:** readString() and writeString().

១. ស្វែងយល់ពី Java 8 - 11 (ត)

Feature in Java 11

- **Developer Feature** មានដូចជា (ត):
  - **Collection to an Array**
  - **The Not Predicate Method**
  - **Local Variable Syntax for Lambda**
  - **HTTP Client**
  - **Nest Best Access Control**
  - **Running Java File**

១. ស្វែងយល់ពី Java 8 - 11 (ត)

Feature in Java 11

- **Performance Enhancement** មានដូចជា:
  - **Dynamic Class-File Constants**
  - **Improved Aarch64 Intrinsics**
  - **A No-Op Garbage Collector**
  - **Flight Recorder**

១. ស្វែងយល់ពី Java 8 - 11 (ត)

Feature in Java 11

- Removed and Deprecated Modules
  - Java EE and CORBA
  - JMC and JavaFX
  - Deprecated Modules
- Miscellaneous Changes

១.១ ស្វែងយល់ពី Functional Programming vs OOP

តើ Functional Programing ជាអ្វី?

- **Function Programming:** គឺជា subset នៃការ declare programming paradigm។ Functional Programming Technique ធ្វើអោយ code ច្បាស់លាស់ Readable, Predictable ជាងមុន។ វាងាយក្នុងការ test និង maintain code ដែល developed តាមរយៈ functional programming។ មួយវិញទៀតវាចូលរួមដោយ concept ដែលសំខាន់ដូចជា Immutable state, referential transparency, method references, high-order និង pure function។ វា involve programming technique ដូចជា functional composite, recursion, currying, functional interface។

១.១ ស្វែងយល់ពី Functional Programming vs OOP

តើ Functional Programing និង OOP មានលក្ខណៈខុសគ្នាដូចម្តេច?

- **Functional Programming (FP)** និង **Object-Oriented Programming (OOP)** គឺជាគំរូពីរផ្សេងគ្នា ហើយ Java 8 បានណែនាំ feature មួយចំនួនដែលជួយសម្រួលដល់ការសរសេរកម្មវិធីដែលមាន functional ជាមួយនឹងសមត្ថភាព OOP ដែលមានស្រាប់របស់វា។ នេះគឺជាការប្រៀបធៀបនៃ FP និង OOP នៅក្នុងបរិបទនៃ Java 8 ទៅ Java 11៖

១.១ ស្វែងយល់ពី Functional Programming vs OOP

- **Immutable Data Structure:**
  - **FP៖** Immutability
    គឺជាគោលគំនិតសំខាន់នៅក្នុង FP ។ Java 8 បានណែនាំ
    `java.util.Optional` class សម្រាប់បង្ហាញ option value
    ដែលធ្វើអោយមិនអាចផ្លាស់ប្តូរបាន(Immutability)។
  - **OOP៖** នៅក្នុង OOP Immutability អាចសម្រេចបានដោយការបង្កើត
    field "final" និងមិនផ្តល់ setters method ទេ។

១.១ ស្វែងយល់ពី Functional Programming vs OOP

- **Higher-order Functions:**
  - **FP៖** Java 8 បានណែនាំ Functional interface និង lambda expressions ដែលអនុញ្ញាតឱ្យប្រើ Higher-order Functions។
  - **OOP៖** ខណៈពេលដែលភាសា OOP អាច support callbacks ឬ function object, ការ implement របស់ Java នៃ Higher-order function កាន់តែល្អជាងមុនជាមួយ lambdas នៅក្នុង Java 8 ។

១.១ ស្វែងយល់ពី Functional Programming vs OOP

- **Streams:**
  - **FP៖** Java 8 បានណែនាំ **`java.util.stream`** package ដោយផ្តល់នូវវិធីដ៏មានអានុភាពដើម្បីដំណើរការ collection នៅក្នុង function style។
  - **OOP៖** នៅពេលដែល OOP ក៏អាចដំណើរការ collection ដោយប្រើ traditional loops និង iterators, Streams ផ្តល់នូវ expressive way និងច្បាស់លាស់ជាងមុន ដើម្បីអនុវត្ត operations នៅលើ collections។

១.១ ស្វែងយល់ពី Functional Programming vs OOP

- **Pure Function:**
  - **FP៖** សង្កត់ធ្ងន់លើ pure function ដែលមិនមានផលប៉ះពាល់ (ការកែប្រែទិន្នន័យ) និង return outputs ដូចគ្នាសម្រាប់ការ inputs ដូចគ្នា។ Function programming នៅក្នុង Java លើកទឹកចិត្តឱ្យសរសេរ pure functions ប៉ុន្តែវាមិនបង្ខំឱ្យធ្វើដាច់ខាតទេ (អាចសរសេរជា pure function ក៏បានអត់ក៏បាន)។
  - **OOP៖ Encapsulation** នៅក្នុង OOP ជួយក្នុងការបង្កើត pure function នៅក្នុង class ប៉ុន្តែ OOP មិនបង្ខំឱ្យធ្វើដាច់ខាតទេ។

១.១ ស្វែងយល់ពី Functional Programming vs OOP

- **State Management:**
  - **FP:** Immutability និង avoidance នៃការចែករំលែក mutable state គឺជា key សំខាន់នៅក្នុង FP ។ Java បានណែនាំ `java.util.concurrent` package ដើម្បី support ការសរសេរ programming ស្របគ្នាជាមួយនឹង mutable state បានតិចតួចបំផុត។
  - **OOP:** ពឹងផ្អែកខ្លាំងលើ mutable state ដែល encapsulated ក្នុង object។ នៅពេលដែល encapsulation ជួយគ្រប់គ្រង state វាក៏អាចនាំឱ្យមាន complex state management នៅក្នុង application ធំៗផងដែរ។

១.១ ស្វែងយល់ពី Functional Programming vs OOP

- **Parallelism:**
  - **FP៖** Java 8 បានណែនាំ parallel streams operations ដែលអនុញ្ញាតឱ្យមាន parallelism កាន់តែងាយស្រួលដោយប្រើប្រាស់ multi-core processors។
  - **OOP៖** នៅពេលដែលភាពស្របគ្នា (parallelism) អាចសម្រេចបាននៅក្នុង OOP តាមរយៈThreading និងការបង្កើត construct parallel stream របស់ Java 8 ផ្តល់នូវវិធីសាស្ត្រ declare កាន់តែច្បាស់ និងមានសក្តានុពលសុវត្ថិភាពជាង។

១.២ ស្វែងយល់ពី Learn new technologies of Java

Learn New technology of Java

- នៅក្នុង Java, technology ថ្មីដែលត្រូវទទួលបាន significant traction គឺជា **Spring Framework** ជាពិសេសកំណែប្រែចុងក្រោយរបស់វាគឺ **Spring boot**។ **Spring boot** simplifies ដំណើរការនៃការ build stand-alone, production-grade Spring-based application។

១.២ ស្វែងយល់ពី Learn new technologies of Java

Learn New technology of Java

- **Spring boot Overview: Spring boot** ចាប់យក opinionated មួយ approach
  ទៅកាន់ Configurations ដោះលែង developer
  ចេញពីត
  មជ្ឈការក្នុងការកំណត់ Boilerplate configuration។ វាផ្តល់ defaults សម្រាប់ configurat
  setup។

១.២ ស្វែងយល់ពី Learn new technologies of Java (ត)

- **Example: Creating a RESTful API with Spring Boot:**
  - **Step 1: Setup Spring Boot Project:** ដំបូងយើងត្រូវការ setup Spring Boot project។ យើងអាចធ្វើបែបនេះបានដោយប្រើ Spring Initializr (https://start.spring.io/) ឬប្រើ build tool ដូចជា Maven ឬ Gradle។
  - **Step 2: Define Dependencies**
  - **Step 3: Create Entity and Repository**
  - **Step 4: Create REST Controller**
  - **Step 5: Run Application**
  - **Step 6: Test API Endpoints**

២. ស្វែងយល់ពី Lambda Expression

អ្វីទៅជាLambda Expression?

- **Lambda Expression:** គឺជាShort Blockនៃកូដដែលទទួលតម្លៃពីParameter ហើយ return តម្លៃត្រឡប់ទៅវិញ។ Lambda ស្រដៀងទៅនឹង Method ដែរ តែមិនបាច់ត្រូវការឈ្មោះ។
- **Syntax:**

```
lambda operator -> body
```

២. ស្វែងយល់ពី Lambda Expression (ត)

- **Lambda Expression Parameter:** មានបីគឺ
  - **Lambda Expression with Zero parameter**

```
() -> System.out.println("Zero parameter lambda");
```

  - **Lambda Expression with Single parameter**

```
(p) -> System.out.println("One parameter: " + p);
```

  - **Lambda Expression with Multiple parameter**

```
(p1, p2) -> System.out.println("Multiple parameters: " + p1
+ ", " + p2);
```

២. ស្វែងយល់ពី Lambda Expression (ត)

Example: Lambda Expression - Single Parameter

```
interface FuncInterface{
    // An abstract function
    void abstractFun(int x);
}
class LambdaTest1{
    public static void main(String args[]){
        // lambda expression to implement above
        FuncInterface fobj = (int x)->System.out.println(5*x);
```

២. ស្វែងយល់ពី Lambda Expression (ភ)

Example: Lambda Expression (ឥ)

```
        // This calls above lambda expression and prints 10.
        fobj.abstractFun(2);
    }
}
```

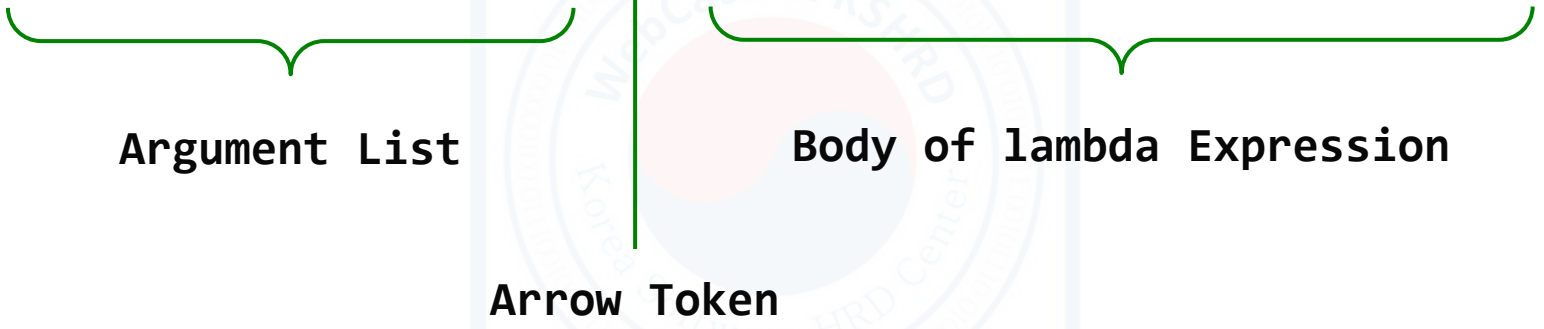
Output

```
10
```

២. ស្វែងយល់ពី Lambda Expression (ត)

Look! Lambda Expression Explicit Argument Type

```
(int arg1, String arg2) -> {System.out.println(arg1 + arg2)
```



KSGA

២.១ ស្វែងយល់ពី Working with Lambda

Example 01: Lambda with Runnable Interface:

```
public class RunnableExample {
    public static void main(String[] args) {
        Runnable runnable = () -> {
            System.out.println("Hello, Lambda!");
        };
        Thread thread = new Thread(runnable);
        thread.start();
    } }
```

Output:

```
Hello, Lambda!
```

២.១ ស្វែងយល់ពី Working with Lambda (ត)

Example 02: Lambda with Functional Interface:

```
interface MyFunctionalInterface {
    int add(int a, int b);
}
public class WithFunctionalInterface {
    public static void main(String[] args) {
        MyFunctionalInterface myFunc = (a, b) -> a + b;
        System.out.println("Result: " + myFunc.add(1, 2));
    }  }
```

Output:

```
Result: 3
```

២.១ ស្វែងយល់ពី Working with Lambda (ត)

Example 03: Lambda with List Iteration:

```
import java.util.Arrays;
import java.util.List;
public class WithListIteration {
    public static void main(String[] args) {
        List<String> myList = Arrays.asList("KSHRD-", "SR");
        myList.forEach(s -> System.out.print(s));
    } }
```

Output:

```
KSHRD-SR
```

២.១ ស្វែងយល់ពី Working with Lambda (ត)

Example 04: Lambda with Comparator-Collection.sort:

```
public class WithComparatorCollection {
    public static void main(String[] args) {
        List<Integer> num = Arrays.asList(1,5,3,4,2);
        Collections.sort(num, (a, b) -> a.compareTo(b));
        System.out.println("Sorted Nums: " + num);
    }    }
```

Output:

```
Sorted Nums: [1, 2, 3, 4, 5]
```

៣. ស្វែងយល់ពី Method References

តើអ្វីទៅជា Method References ?

- **Method References** គឺប្រើវាដោយយោងតាម Method នៃមុខងារ Interface។ វាជាប្រភេទពិសេសនៃកន្សោម Lambda។

៣. ស្វែងយល់ពី Method References(ត)

❖ Type of Method References

Type	Syntax
Static Method References	ContainingClass::staticMethodName
Instance Method Reference of a particular Object	containingObject::instanceMethodName

៣. ស្វែងយល់ពី Method References(ត)

❖ Type of Method Reference(ក)

Type	Syntax
Instance Method Reference of an arbitrary Object of a particular Object Type	ContainingType::methodName
Constructor Method	ClassName::new

៣. ស្វែងយល់ពី Method References(ត)

- **Example:** Static Method References

```
interface Sayable{
    void say();
}
public class MethodReference {
    public static void saySomething(){
        System.out.println("Hello, this is static method.");
}
```

៣. ស្វែងយល់ពី Method References(ត)

- **Example:** Static Method References(ក)

```
    public static void main(String[] args) {
            // Referring static method
            Sayable sayable = MethodReference::saySomething;
            // Calling interface method
            sayable.say();
        }
    }
```

៣. ស្វែងយល់ពី Method References(ត)

- **Example:** Static Method References(ត)

Output

```
Hello, this is static method.
```

៣. ស្វែងយល់ពី Method References(ត)

❖ **Example:** Instance Method Reference of a particular Object

```
interface Sayable{
    void say();
}
public class InstanceMethodReference {
    public void saySomething(){
        System.out.println("Hello, this is non-static method.");
    }
```

៣. ស្វែងយល់ពី Method References(ត)

❖ **Example:** Instance Method Reference of a particular Object(ព)

```
public static void main(String[] args) {
    InstanceMethodReference methodReference = new
InstanceMethodReference(); // Creating object
            // Referring non-static method using reference
                Sayable sayable = methodReference::saySomething;
            // Calling interface method
                sayable.say();
```

៣. ស្វែងយល់ពី Method References(ត)

❖ **Example:** Instance Method Reference of a particular Object(ព)

```
// Referring non-static method using anonymous object
Sayable sayable2 = new InstanceMethodReference()::saySomething;
// You can use anonymous object also
// Calling interface method
        sayable2.say();
        }
    }
```

៣. ស្វែងយល់ពី Method References(ត)

- **Example:** Instance Method Reference of a particular Object(ព)

Output

```
Hello, this is non-static method.
Hello, this is non-static method.
```

៣. ស្វែងយល់ពី Method References(ត)

- **Example:** Instance Method Reference of an arbitrary Object of a particular Object Type

```
import java.util.*;
public class Method_Instance {
   public static void main(String[] args)
   {
       // Creating an empty ArrayList of user defined type
       List<String> personList = new ArrayList<>();
```

៣. ស្វែងយល់ពី Method References(ត)

- **Example:** Instance Method Reference of an arbitrary Object of a particular Object Type(ត)

```
        personList.add("kinin");
        personList.add("manin");
        personList.add("ryta");
        // Method reference to String type
        Collections.sort(personList,
                String::compareToIgnoreCase);
        personList.forEach(System.out::println);
    }}
```

៣. ស្វែងយល់ពី Method References(ត)

- **Example:** Instance Method Reference of an arbitrary Object of a particular Object Type(ត)

Output

kinin

manin

ryta

៣. ស្វែងយល់ពី Method References(ត)

- **Example:** Constructor Method

```
interface Messageable{
    Message getMessage(String msg);
}
class Message{
    Message(String msg){
        System.out.print(msg);
    } }
```

៣. ស្វែងយល់ពី Method References(ត)

- **Example:** Constructor Method(ក)

```
public class ConstructorReference {

    public static void main(String[] args) {

        Messageable hello = Message::new;

        hello.getMessage("I Love JAVA");

    }

}
```

៣. ស្វែងយល់ពី Method References(ត)

- **Example:** Constructor Method(ត)

Output

I Love JAVA

៤. ស្វែងយល់ពី Functional Interface

- **Functional Interface** គឺជា interface ដែល contains វាមានតែ abstract method មួយតែប៉ុណ្ណោះ។ function Interface អាច បន្ថែម recognized Single Abstract Method Interface (SAM Interface)។
- **Functional Interface** វា include in java SE 8 ជាមួយនិង Lambda expression ហើងនិង Method references សម្រាប់ធ្វើការ order straightforward and clean។

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Functional Interface** ប្រើសម្រាប់ការ executed by representing interface ដែល annotation ហៅថា " **@FunctionalInterface** "។
- **Functional Interface** វាមិនការប្រើ abstract keyword ដោយសារវា by default។ Method defined ក្នុង interface បានទេ abstract មួយតែប៉ុណ្ណោះ ហើយយើងអាច call Lambda expressions instance of functional interface។
- **@FunctionalInterface** វា ensure ថា interface មិនអាចមាន abstract method លើសពីមួយទេ។

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Example:** assign lambda expression

```
class Test {
   public static void main(String args[]){
   //lambda expression to create the object
        new Thread(() -> {
            System.out.println("New thread created");
        }).start();}}
```

Output

```
New thread created
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Example:** build functional interface

```
@FunctionalInterface
public interface MyfristFunctionalInterface {
   void firstWork();
}
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Example:** when try to add another abstract method

```
@FunctionalInterface
public interface MyfristFunctionalInterface {
    void firstWork();
    void doSomeMoreWork();//Error
}
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Since java SE 1.8** មាន interface ជាច្រើន converted into function interface។ ហើយinterfaces ទាំងអស់នោះប្រើ annotated with @FunctionalInterface ដែលមាន interface ដូចជា ៖
  - Runnable -> only contains the run() method.
  - Comparable -> only contains the compareTo() method.
  - ActionListener -> only contains the actionPerformed() method.
  - Callable-> only contains the call() method.

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Java 8 functional interfaces** វាអាច applied multiple situations ដែលមានដូចជា៖
  - **Consumer**
  - **Predicate**
  - **Function**
  - **Supplier**

ហើយក្នុងចំណោម 4 ខាងលើ មាន 3 interface ដូចជា Consumer, Predicate and Function ដែល additions that are provided beneath –

៤. ស្វែងយល់ពី Functional Interface(ត)

1. Consumer -> Bi-Consumer
2. Predicate -> Bi-Predicate
3. Function -> Bi- Function, Unary Operator, Binary Operator

- **Consumer interface** ជា inference ដែល accept only one argument ហើយវាមិន return value ទេ។
- **Syntax:**

```
public interface Consumer<T> {
        void accept(T t);
}
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Example:** Consumer

```
public class Person {
    private String name;
    public String getName() {
        return name;}
    public void setName(String name) {
        this.name = name;}
}
```

៤. ស្វែងយល់ពី Functional Interface(ត)

❖ **Example:** Consumer

```
import java.util.function.Consumer;
public class Main {
   public static void main(String[] args) {
        Person p=new Person();
        Consumer<Person> consumer=t ->
t.setName("play java");
        consumer.accept(p);
        System.out.println(p.getName());}}
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Output:**

```
play java
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Predicate:** វា accepts an argument និង generates a boolean value។
- **Syntax:**

```
public interface Predicate<T> {
    Boolean test(T t);
}
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Example:** Predicate

```
import java.util.function.Predicate;
public class Main {
    public static void main(String[] args) {
      Predicate<String>checklenght=str->str.length()>5;
      System.out.println(checklenght.test("checklenght"));}}
```

Output

```
true
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Function**: វាទទួលទាំង an argument ផង និង result ផង។
- **Syntax:**

```
public interface Function<T,U,R> {
    R apply(T t,U u);
}
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Example:** Function

```
import java.util.function.Function;
public class Main {
   public static void main(String[] args) {
       Function<Integer,String> getInt=t->t*10+"data
multiplied by 10";
       System.out.println(getInt.apply(2));}}
```

Output:

```
20data multiplied by 10
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Supplier:** គឺវាមិនទទួលការ input or argument ហើយនៅតែ return a single output។
- **Syntax:**

```
public interface Supplier<T> {
    T.get();
}
```

៤. ស្វែងយល់ពី Functional Interface(ត)

- **Example:** Supplier

```
import java.util.function.Supplier;
public class Main {
    public static void main(String[] args) {
        Supplier<Double> getRandomDoble=()->Math.random();
        System.out.println(getRandomDoble.get());
    }}
```

Output:

```
0.7367699679370314
```

៥. ស្វែងយល់ពី Default Method

អ្វីទៅជាDefault Method?

- **Default Method:** គឺប្រើប្រាស់នូវ default keyword នៅពីមុខ abstract method នៅក្នុងInterface។ យើងបានដឹងរួចហើយថា interface មិនអាចមាន method ដែលមាន body និង class ដែល Implements ពីវា ត្រូវតែ @override methods ទាំងនោះ។
- **ចាប់ពីJava 8** ឡើងទៅ ដោះស្រាយបញ្ហាខាងលើ យើងអាចបង្កើត method ដែលមាន body នៅក្នុង interfaceបាន ហើយនៅពេល subclass implements
  គឺមិនត្រូវការ @Override នោះទេ។

៥. ស្វែងយល់ពី Default Method (ត)

- **Syntax:**

```
interface Interface_Name {
    //default method
    default void method_name() {
        //body
    }
}
```

៥. ស្វែងយល់ពី Default Method (ត)

Example: Default Method and Abstract Method (ត)

```
class Test implements TestInterface
{
   // implementation of square abstract method
   @Override
   public void square(int a) {
        System.out.println(a+a);
   }
```

៥. ស្វែងយល់ពី Default Method (ត)

Example: Default Method and Abstract Method (ព)

```
    public static void main(String args[]){
        Test d = new Test();
        d.square(5);
        // default method executed
        d.show();
    }  }
```

Output:

```
10
Default Method Executed
```

៥. ស្វែងយល់ពី Default Method (ត)

Example: Default Method and Abstract Method

```
interface TestInterface{
    // abstract method
    void square(int a);
    // default method
    default void show(){
        System.out.println("Default Method Executed");
    }
}
```

៦. ស្វែងយល់ពី New Date Time API

ស្វែងយល់ពី New Date Time API

- **New Date Time API** គឺជាការណែនាំរបស់Java 8 ដើម្បី overcome ទៅលើបញ្ហានៃ Date Time API ។
  - **Not Thread Safe:** មិនដូច java.util.Date ចាស់ ហើយអ្វីដែរមិនមានសុវត្ថិភាពនោះគឺ New Date Time API គឺមិនអាចប្រែប្រួលបាន និងមិនមាន setter methods ទេ។
  - **Less Operations:** នៅក្នុង Old API មានប្រតិបត្តិការ Date តិចតួចប៉ុណ្ណោះប៉ុន្តែ New API ផ្តល់ឱ្យយើងនូវប្រតិបត្តិការកាល Date ជាច្រើន។

៦. ស្វែងយល់ពី New Date Time API (ត)

- **JAVA 8** under the package java.time បានណែនាំ new date time API ។ Class សំខាន់បំផុតក្នុងចំណោមពួកគេគឺ

1. **Local :** date time API ចាប់យកនូវ Local Date time current នៅក្នុងតំបន់។
2. **Zoned:** date time API ចាប់យកនូវ timezoned តំបន់ផ្សេងទៀត។

- **Key features of the Java 8 Date-Time API**
  - **LocalDate/LocalTime** and **LocalDateTime API :** Use it when time zones are NOT required.

៦. ស្វែងយល់ពី New Date Time API(ត)

- **Example: LocalDateTime API**

```
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.Month;
public class Local_Date_time {
    public static void main(String[] args) {
        LocalDateTime currentTime = LocalDateTime.now();
        System.out.println("Current DateTime: " + currentTime);
```

៦. ស្វែងយល់ពី New Date Time API(៣)

- **Example: LocalDateTime API (៥)**

```
LocalDate date = currentTime.toLocalDate();
System.out.println("date: " + date);
Month month = currentTime.getMonth();
int day = currentTime.getDayOfMonth();
int hour = currentTime.getHour();
int min = currentTime.getMinute();
int sec = currentTime.getSecond();
```

៦. ស្វែងយល់ពី New Date Time API(ត)

- **Example: LocalDateTime API (ត)**

```
 System.out.println("Month: " + month +" - Day: " + day +"\nHour : "+
hour + ", Minute: " + min + ", Second: " + sec);
    }
}
```

៦. ស្វែងយល់ពី New Date Time API(៥)

- **Example: LocalDateTime API(៥)**

Output

Current DateTime: 2024-02-25T19:29:07.835706800

date: 2024-02-25

Month: FEBRUARY - Day: 25

Hour : 19, Minute: 29, Second: 7

៦. ស្វែងយល់ពី New Date Time API(ត)

➢ **Zoned date-time API** : Use it when time zones are to be considered.

❖ **Example: Zoned date-time API**

```
import java.time.ZoneId;
import java.time.ZonedDateTime;
public class Zoned_Date_Time {
    public static void main(String[] args) {
        ZoneId currentZone = ZoneId.systemDefault();
        System.out.println("Current Zone: "+
ZonedDateTime.now().getZone());
```

៦. ស្វែងយល់ពី New Date Time API(៥)

Example: Zoned date-time API (៥)

```
 ZonedDateTime zoneDate = ZonedDateTime.parse(
"2007-12-03T10:15:30+05:30[Europe/Paris]");
      System.out.println("zoneDate: " + zoneDate.getZone());
   }
}
```

៦. ស្វែងយល់ពី New Date Time API(៣)

Example: Zoned date-time API (៣)

Output

Current Zone: Asia/Bangkok

zoneDate: Europe/Paris

៦. ស្វែងយល់ពី New Date Time API(ត)

- **Chronologies:** is added in Java 8 to replace integer values used in old API to represent day, month etc.
- **Example:** Chronologies

```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
public class Chronologies {
   public static void main(String[] args) {
       LocalDate today = LocalDate.now();
       System.out.println("Current date: " + today);
```

៦. ស្វែងយល់ពី New Date Time API(ត)

- **Example:** Chronologies(ត)

```
//add 1 week to the current date
LocalDate nextWeek = today.plus(1, ChronoUnit.WEEKS);
System.out.println("Next Week: " + nextWeek);
//add 1 month to the current date
LocalDate nextMonth = today.plus(1, ChronoUnit.MONTHS);
System.out.println("Next Month: " + nextMonth);
//add 1 year to the current date
LocalDate nextYear = today.plus(1, ChronoUnit.YEARS);
System.out.println("Next Year: " + nextYear);
```

៦. ស្វែងយល់ពី New Date Time API(៥)

- **Example:** Chronologies(៥)

```
        //add 10 years to the current date
        LocalDate nextDecade = today.plus(1, ChronoUnit.DECADES);
        System.out.println("Date after 10 year: " + nextDecade);
    }
}
```

៦. ស្វែងយល់ពី New Date Time API(ត)

- **Example:** Chronologies(ត)

Output

Current date: 2024-02-25

Next Week: 2024-03-03

Next Month: 2024-03-25

Next Year: 2025-02-25

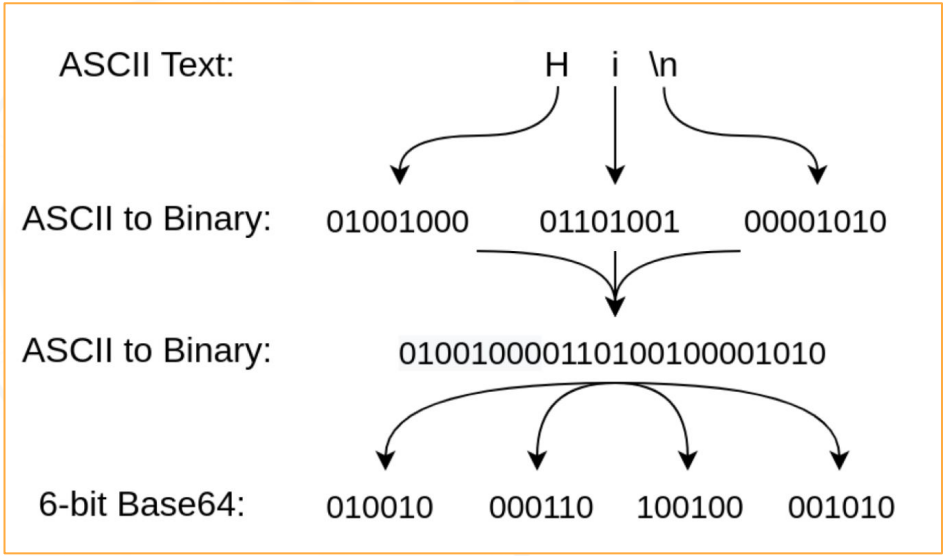
Date after 10 year: 2034-02-25

៧. ស្វែងយល់ពី Base64

- **អ្វីទៅជាBase64?៖** នៅក្នុង Java 8 គេបានបន្ថែមនូវ Base64 ដែលវាជា Binary-to-text encoding scheme ដែលនិយាយពីការ Converts binary data ទៅជា ASCII String format។ ហើយវាត្រូវបានប្រើប្រាស់ដើម្បី encode binary data ដូចជា URL និង អក្សរ ផ្សេងៗ អោយទៅជា **“អក្សរ”** មួយទៀតដែលតំណាងឱ្យ**“URL”**និង**“អក្សរ”**ទាំងនោះ ហ្នឹងអោយទៅជាអក្សរមួយផ្សេងទៀត។
- **តើវាធ្វើការដោយរបៀបណា?៖** Base64 វាធ្វើការ transforms binary data ទៅជា Textual format ដែលវាមានជាមួយ a set of 64 ASCII (A-Z, a-z, 0-9, '+', និង '/') ។ហើយវាធ្វើការបំបែក binary data ម្តង 6-bit និងភ្ជាប់វាទៅកាន់ អក្សរដែលត្រូវហ្នឹងវា។

៧. ស្វែងយល់ពី Base64 (ត)

- តើវាធ្វើការដោយរបៀបណា?៖ (ត)



KSGA

៧. ស្វែងយល់ពី Base64 (ត)

៧.១ ស្វែងយល់ពី How to Encrypt Content with base64

- **របៀប Encrypt Content ជាមួយ Base64៖** ជាចំណុចសំខាន់ត្រូវចាំ Base64 មិនមែន Encryption Algorithm តែវាជា Encoding Scheme ដោយសារវាមិនបានផ្តល់នៅ Security ផ្សេងៗប៉ុន្តែត្រូវបានប្រើយ៉ាងទូលំទូលាយទៅលើការ Encode Data សម្រាប់ការបញ្ជូនឬការផ្ទុកក្នុងទម្រង់ដែលមានសុវត្ថិភាព.

៧. ស្វែងយល់ពី Base64 (ត)

៧.១ ស្វែងយល់ពី How to Encrypt Content with base64

- **Example:**

```
import java.util.Base64;
public class Base64Example {
        // Original content to be encoded
        public static void main(String[] args) {
            // Original content to be encoded
            String originalContent = "Hello, Base64 Encoding!";
```

៧. ស្វែងយល់ពី Base64 (ត)

៧.១ ស្វែងយល់ពី How to Encrypt Content with base64

- **Example: (ក)**

```
// Encoding content to Base64
String encodedContent =
Base64.getEncoder().encodeToString(originalContent.getBytes());
System.out.println("Encoded Content: " + encodedContent);
```

៧. ស្វែងយល់ពី Base64 (ត)

៧.១ ស្វែងយល់ពី How to Encrypt Content with base64

- **Example: (ត)**

```
// Decoding Base64 content
            byte[] decodedBytes =
Base64.getDecoder().decode(encodedContent);
            String decodedContent = new String(decodedBytes);
            System.out.println("Decoded Content: " + decodedContent);
        }
}
```

៧. ស្វែងយល់ពី Base64 (ត)

៧.១ ស្វែងយល់ពី How to Encrypt Content with base64

- **Example: (ត)**

```
Output:
        Encoded Content: SGVsbG8sIEJhc2U2NCBFbmNvZGluZyE=
        Decoded Content: Hello, Base64 Encoding!
```

៨. ស្វែងយល់ពី Stream API

- **Stream:** តំណាងអោយលំដាប់នៃobject ពីsource។ដែលsupports aggregate operations។
- តាំងពី**Java 8** មក , java បានផ្តល់នូវ new stream API ដើម្បីធ្វើការជាមួយstreamដើម្បី filter, collect, printនិង convert ពីsource ដូចជាdata structure, array រឺ I/O channel តាមរយៈ pipeline នៃ computational operation។

៨. ស្វែងយល់ពី Stream API (ត)

- ដើម្បីបង្កើតនូវEmpty Stream យើងអាចប្រើ empty() method:

Example:

```
Stream<String> streamEmpty = Stream.empty();
```

៨. ស្វែងយល់ពី Stream API (៣)

- ដើម្បីបង្កើតstream នៃប្រភេទនៃcollection(collection, List , Set) គេប្រើ stream() method

  Example:

```
collection<String> collection = Arrays.asList("a", "b", "c");
    Stream<String> streamOfCollection = collection.stream();
```

៨. ស្វែងយល់ពី Stream API (ត)

- យើងក៏អាចបង្កើតstream នៃArray ដោយប្រើប្រាស់ Stream.of(Array[]) method

Example:

```
Stream<String> streamOfArray = Stream.of("a", "b","c");
```

៨. ស្វែងយល់ពី Stream API (ត)

Example: Filtering Collection by using Stream

```
import java.util.*;
import java.util.stream.Collectors;
class Product{
    int id;
    String name;
    float price;
```

៨. ស្វែងយល់ពី Stream API (៥)

Example: Filtering Collection by using Stream(៥)

```
    public Product(int id, String name, float price) {
            this.id = id;
            this.name = name;
            this.price = price;
        }
    }
```

៨. ស្វែងយល់ពី Stream API (ត)

Example: Filtering Collection by using Stream(ត)

```
public class Main {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>(){{
            //Adding Products
            add(new Product(1,"HP Laptop",25000f));
            add(new Product(2,"Dell Laptop",30000f));
            add(new Product(3,"Lenevo Laptop",28000f));
```

៨. ស្វែងយល់ពី Stream API (ត)

Example: Filtering Collection by using Stream(ត)

```
        add(new Product(4,"Sony Laptop",28000f));
        add(new Product(5,"Apple Laptop",90000f));
    }};
```

៨. ស្វែងយល់ពី Stream API (៥)

Example: Filtering Collection by using Stream(៥)

```
List<Float> productPriceList2 =productsList.stream()
                .filter(p -> p.price > 30000)// filtering data
                .map(p->p.price)        // fetching price
                .collect(Collectors.toList()); // collecting as list
        System.out.println(productPriceList2);
    }
}
```

៨. ស្វែងយល់ពី Stream API (៣)

Output:

```
[90000.0]
```

៩. ស្វែងយល់ពី forEach() Method

- **អ្វីទៅជា forEach() method?៖** forEach() method ត្រូវ Introduced in Java 8 ហើយវាជា part of the Stream API ដែល។
  វាត្រូវបានប្រើប្រាស់ដើម
  ្បី Iterating ឆ្លងកាត់ item នីមួយៗ ក្នុង Collection ដូចជា list, Set ឬក៏ elements in St
  - **សំរាប់ Collection៖** ប្រសិនបើយើងមាន a Set of items (ដូចជាname in a list) “forEach()” វាជួយយើងប្រើប្រាស់វាស្រួយជាងមុនដោយមិនចាំបាច់ប្រើ **traditional loop**

៩. ស្វែងយល់ពី forEach() Method (ត)

- **សំរាប់ Stream**៖ ប្រសិនបើយើងចង់ធ្វើការជាមួយ Stream នៃ data ដូចជា sequence of numbers forEach ជួយយើង perform an action រាល់ element នីមួយៗនៅក្នុង stream។ហើយយើងត្រូវតែប្រើប្រាស់នូវ **“Lambda expression”**។
- **សំរាប់ Map**៖ ការប្រើប្រាសើជាមួយ Maps វាជួយយើងនូវការងាយស្រួលរក key-Value pair នីមួយៗ។

៩. ស្វែងយល់ពី forEach() Method (ត)

- **Example:**

```
import java.util.*;
public class foreachMethod {
        public static void main(String[] args) {
            List<String> namesList = new ArrayList<>();
            namesList.add("Alice");
            namesList.add("Bob");
            namesList.add("Charlie");
            System.out.println("Names in List:");
            namesList.forEach(name -> System.out.println("Name: " + name));
```

៩. ស្វែងយល់ពី forEach() Method (ត)

- **Example: (ត)**

```
// Example with Set
Set<Integer> numbersSet = new HashSet<>();
numbersSet.add(1);
numbersSet.add(2);
numbersSet.add(3);
numbersSet.add(4);
System.out.println("Numbers in Set:");
numbersSet.forEach(number -> System.out.println("Number: " +
number));
```

៩. ស្វែងយល់ពី forEach() Method (ត)

- **Example: (ក)**

```
// Example with Map
Map<Integer, String> studentMap = new HashMap<>();
studentMap.put(1, "Alice");
studentMap.put(2, "Bob");
studentMap.put(3, "Charlie");
System.out.println("Map:");
studentMap.forEach((key, value) -> System.out.println("Key: " + key
+ ", Value: " + value));
```

៩. ស្វែងយល់ពី forEach() Method (ត)

- **Example: (ត)**

```
// Example with Stream
                List<Integer> numbers = Arrays.asList(1, 2, 3, 4);
                System.out.println("Stream:");
                numbers.stream().forEach(number ->
System.out.println("Number: " + number));
     }
}
```

៩. ស្វែងយល់ពី forEach() Method (ត)

- **Output:**

```
Names in List:
Name: Alice
Name: Bob
Name: Charlie
Numbers in Set:
Number: 1
Number: 2
Number: 3
Number: 4
```

```
Map:
Key: 1, Value: Alice
Key: 2, Value: Bob
Key: 3, Value: Charlie
Stream:
Number: 1
Number: 2
Number: 3
Number: 4
```

១០.១ ស្វែងយល់ពី Optional

តើអ្វីទៅជា Optional ?

- **Optional class:** ត្រូវបានបន្ថែមនៅJava 8, វាគឺជា public final class និងប្រើដើម្បីធ្វើការជាមួយ NullPointerException ក្នុងjava application។ ហើយយើងត្រូវ import java.util package ដើម្បីប្រើក្នុងclass ។ វាក៏បានផ្តល់នូវ method ដែលប្រើដើម្បីcheck presence នៃតម្លៃសម្រាប់particular variable។

១០.១ ស្វែងយល់ពី Optional

- វាបានផ្តល់ alternate mechanism ប្រសើរជាងមុនសម្រាប់method ដើម្បីបង្ហាញថាមិនមានលទ្ធផលទៅអ្នកហៅ។

  Example:

```
Optional<ProductDto> selectById(Integer id);
```

១០.១ ស្វែងយល់ពី Optional (ត)

- ការប្រើ optional class នឹងផ្តល់ការងាយស្រួលដើម្បីប្រើ throw exception

Example:

```
productDao.selectById(12).orElseThrow(() ->
                    new RuntimeException(""));
```

១០.២ Why use Optional instead of try-catch, if-else

Why use Optional instead of try-catch , if-else?

- បានជាគេប្រើoptional ជំនួសtry-catch , if-else ព្រោះ៖
  - Expressive Intent: ធ្វើអោយcode របស់យើងគឺexpressive ជាមុន។
  - Avoidance of Null References: optional វាជួយការពារ NullPointerException ដែលបង្ហាញយ៉ាងច្បាស់នៅពេលតម្លៃមិនមាន។
  - Readability and Maintainability: optional ជាញឹកញាប់វាគឺមានលទ្ធផលច្បាស់ បើប្រៀបជាមួយif-else and try-catch។

១០.២ Why use Optional instead of try-catch, if-else (ត)

- Functional Programing : Optional វាល្អជាមួយនឹងfunctional programing style។
- Chaining and Composition : optional វាផ្តល់នៅmethod សម្រាប់chaining operation ដូចជា map , flatMap, filter។
- Error Handling vs Absence Handling: try-catch ជាទូទៅប្រើសម្រាប់handling exceptional condition។
- Performance: optional ជាធម្មតាវាល្អប្រសើរជាងបើប្រៀបធៀបជាមួយការប្រើexception សម្រាប់control flow។

១០.៣ ស្វែងយល់ពី How to use Optional

How to use Optional ?

- Import the java.util.Optional class:

```
import java.util.Optional;
```

- បង្កើតOptional variable and set its value using the of or ofNullable method.

១០.៣ ស្វែងយល់ពី How to use Optional (ត)

- ប្រើពេលយើងប្រាកដថាតម្លៃមិនnull

```
Optional<String> optionalValue = Optional.of("Non-null value");
```

- ប្រើofNullable នៅពេលតម្លៃអាចនឹងnull

```
String nullableValue = null;
Optional<String> optionalValue =
Optional.ofNullable(nullableValue);
```

១០.៣ ស្វែងយល់ពី How to use Optional (ត)

- ដើម្បីទទួលតម្លៃពីoptional ប្រើget method

```
if (optionalValue.isPresent()) {
        String value = optionalValue.get();
        // Use the value
        } else {
        // Handle the case when the value is absent
        }
```

១០.៣ ស្វែងយល់ពី How to use Optional (ត)

- ដើម្បីទទួលតម្លៃពីoptional ប្រើget method(ត)
- Or, you can use the orElse method to provide a default value:

```
String value = optionalValue.orElse("Default value");
```

១០.៣ ស្វែងយល់ពី How to use Optional (ត)

- យើងអាចប្រើប្រាស់នូវ method ifPresent, Else Throw , filter ដើម្បីប្រើជាមួយ optional value

Example:

```
optionalValue.ifPresent(value -> {
        // Perform an action with the value
        });
        try {
        String value = optionalValue.orElseThrow();
        // Use the value }
```

១០.៣ ស្វែងយល់ពី How to use Optional (ត)

Example: (ត)

```
        catch (NoSuchElementException e) {
            // Handle the case when the value is absent
            }
            Optional<String> filteredOptional =
        optionalValue.filter(value -> value.length() > 5);
        filteredOptional.ifPresent(value -> {
        // Perform an action with the filtered value
        });
```

១០.៤ ស្វែងយល់ពី using Optional

គោលបំណង់នៃការប្រើ **optional <T>** នៅក្នុង **Java** ៖

- **Optional** គឺជា class មួយដែលតំណាងឲ្យ Presence or absence នៃអ្វីមួយ។
- **Optional** វាគឺជា **Wrapper class** សម្រាប់ **Generic type** ។
- **Example**: Optional instance is empty ប្រសិនបើ T នេះវា Null។

ក្នុង **Java 11** គោលបំណងវាគឺផ្តល់នូវ return type ដែលអាចតំណាងឲ្យ Presence the absence value ក្នុង scenarios ដែល return Null ដែលអាចធ្វើឲ្យមានកំហុស error ដែលមិនបានរំពឹងទុក ដូចជា NullPointerException។

១០.៥ ស្វែងយល់ពី Optional Methods

- **Optional Method class** វាផ្តល់ប្រយោជន៏ ឲ្យយើងធ្វើការជាមួយ API ។
- **Methode** ដែលសំខាន់ៗមានដូចជា៖
  - **of():** សម្រាប់ return instance នៃ optional ជាមួយ value ខាងក្នុង
  - **orElse():** សម្រាប់ return value ខាងក្នុង optional otherwise return optional ផ្សេងទៀត។
  - **employ():** សម្រាប់ return an empty instance នៃ optional។

៣. ឯកសារយោង

- https://www.baeldung.com/new-java-9
- https://www.javatpoint.com/java-lambda-expressions
- https://www.geeksforgeeks.org/are-all-methods-in-a-java-interface-are-abstract/
- https://www.baeldung.com/java-10-overview
- https://www.digitalocean.com/community/tutorials/java-10-features
- https://www.geeksforgeeks.org/java-8-features/
- https://www.baeldung.com/java-optional-uses
- https://www.baeldung.com/java-11-new-features

៣. ឯកសារយោង

- https://www.javatpoint.com/java-8-method-reference
- https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html
- https://www.geeksforgeeks.org/new-date-time-api-java8/?ref=lbp
- https://medium.com/@moiz.mhb/new-java-8-date-time-api-155249df3e8
- https://www.javatpoint.com/java-8-stream

មជ្ឈមណ្ឌលកូរ៉េ សហ្វវែរ អេច អ ឌី

Korea Software HRD Center

មូលនិធិ សហ្វវែរ កូរ៉េ សម្រាប់ជំនួយជាសកល

Foundation for Korea Software Global Aid

សូមអរគុណ!