Vietnam National University of HCMC
International University
School of Computer Science and Engineering

Course Revision

(IT069IU)

Le Duy Tan, Ph.D.

ldtan@hcmiu.edu.vn

leduytanit.com

Previously,

- **What** is Web Scraping?
- **Why** is it important?
- **How** to use it safely and legally?
- HTML, CSS
- Introduction to **JSoup** Library
- JSoup workflow
- VNExpress Scrapping Demo
- Manga Scrapping Demo
- Potential issues with Web Scraping

Agenda

- Course revision:
  - Go through topics we have learnt for each week
- Final exam format:
  - Number of questions
  - Topics
  - Time
- Final Game Project
- Final Thought & Message of Tom

Lecture 1

Intro to OOP

Lecture 1 - Intro to OOP

- What are Programming Paradigms?
- What is Object Oriented Programming (OOP) ?
- Why should we learn Java?
- OOP Concepts
  - Class
  - Object
  - Encapsulation with Access Modifier
  - Abstraction
  - Inheritance
  - Composition
- What are JDK, JRE, JVM? How these works together?

Programming Paradigms

- Different approaches to build solutions to specific problems.
- Most programming languages belong to one paradigm.
- Two most popular programming paradigms:
  - Procedural Programming
  - Object-Oriented Programming

Java is still the king of job market in Vietnam

Attractive Salaries in Vietnam

Senior Java Developers
2,000 - 2,300 USD

Mid-Senior Java Dev (Spring, SQL, HTML)
1,500 - 2,500 USD

Backend Developer (Java/Spring)
1,200 - 3,500 USD

Java Developer (Spring, MySQL)
1,000 - 2,500 USD

The most popular language at work in Vietnam

(Vietnam Developer Report 2019)

Nearly 1 out of every 5 IT jobs is Java

Java Job Market

ITViec.com

How Java works

=> Java is **cross platform** - “Write once, run everywhere!”

Lecture 2

Intro to Java

Lecture 2 - Intro to Java

- Different Java Platforms
- Our choice of JDK
- The best IDE selections
- Create the first Java programs
- Compile and Run with commands or on IntelliJ
- Java data types:
  - Primitive
  - Non-primitive (reference types)
- Variable
- Operators

Our first Java Program

MyFirstJavaProgram.java

```java
public class MyFirstJavaProgram {
    public static void main(String[] args) {

        // This will print out the string "Hello World"
        System.out.println("Hello World");

    }
}
```

Data Types

- Java is statically typed and also a strongly typed language.
- There are two categories of data types:
  - **Primitive** data types (8)
  - **Non-primitive** data types (**reference** type)

Operators

All the **Java operators** can be divided into the following groups –

- **Arithmetic** Operators: * / % + -
- **Relational** Operators: < > <= >= == != instanceof
- **Bitwise** Operators: & ^ |
- **Logical** Operators: && || !
- **Assignment** Operators: = += -= *= /=
- **Ternary** operator: ? :

Lecture 3

Intro to Classes & Objects in Java

Lecture 3 - Intro to Classes & Objects in Java

- **Useful Classes**
  - **Scanner**
    - Read input string with nextLine(), next()
    - Read input number with nextDouble()
  - **String**
    - Display string with print(), println(), printf()
  - **Math**
    - pow(), max(), random()
- **Primitive** vs **Reference**
- **Class**
  - **Attributes**
  - **Method** with Parameters
    - **Getters** and **Setters** Methods
  - **Access Modifiers** (Public & Private)
  - **Constructor** with Parameters
  - **UML** Diagram
- **Object**
  - **Create objects** from class with keyword **new**
  - Call **methods** with input arguments
- Bank account application example

Scanner Class: Reading Inputs

- API documentation (http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html)
- To use Scanner class, remember to **import java.util.Scanner**
- Can read from different **input sources**: input keyboards, files, …
- Can convert inputs to different type of **data types**: string, byte, short, int, float, double, …

Java naming convention Method names – lowerCamelCase

String	`next(String pattern)` Returns the next token if it matches the pattern constructed from the specified string.
BigDecimal	`nextBigDecimal()` Scans the next token of the input as a `BigDecimal`.
BigInteger	`nextBigInteger()` Scans the next token of the input as a `BigInteger`.
BigInteger	`nextBigInteger(int radix)` Scans the next token of the input as a `BigInteger`.
boolean	`nextBoolean()` Scans the next token of the input into a boolean value and returns that value.
byte	`nextByte()` Scans the next token of the input as a `byte`.
byte	`nextByte(int radix)` Scans the next token of the input as a `byte`.
double	`nextDouble()` Scans the next token of the input as a `double`.
float	`nextFloat()` Scans the next token of the input as a `float`.
int	`nextInt()` Scans the next token of the input as an `int`.
int	`nextInt(int radix)` Scans the next token of the input as an `int`.
String	`nextLine()`

String Class: Representation in Text

- API documentation (http://docs.oracle.com/javase/7/docs/api/java/lang/String.html)

charAt()
concat()
equals()
indexOf()
lastIndexOf()
length()
toLowerCase()
toUpperCase()
substring()
trim()

And many more…

int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.
int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
String	**intern**() Returns a canonical representation for the string object.
boolean	**isEmpty**() Returns true if, and only if, **length()** is 0.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(String str) Returns the index within this string of the last occurrence of the specified substring.
int	**lastIndexOf**(String str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
boolean	**matches**(String regex)

Math Class: Performing Computation

- API documentation (http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html)
- 2 class attributes:
  - E
  - PI

abs()
ceil()
floor()
hypot()
max()
min()
pow()
random()
sqrt()

And many more…

static double	**abs**(double a) Returns the absolute value of a double value.
static float	**abs**(float a) Returns the absolute value of a float value.
static int	**abs**(int a) Returns the absolute value of an int value.
static long	**abs**(long a) Returns the absolute value of a long value.
static double	**acos**(double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.
static double	**asin**(double a) Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan2**(double y, double x) Returns the angle *theta* from the conversion of rectangular coordinates (x, y) to polar coordinates (r, *theta*).
static double	**cbrt**(double a) Returns the cube root of a double value.
static double	**ceil**(double a) Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.
static double	**copySign**(double magnitude, double sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static float	**copySign**(float magnitude, float sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static double	**cos**(double a) Returns the trigonometric cosine of an angle.
static double	**cosh**(double x) Returns the hyperbolic cosine of a double value.
static double	**exp**(double a) Returns Euler's number *e* raised to the power of a double value.

Primitives vs References

Primitive types are basic Java types

- int, long, double, boolean, char, short, byte, float.
- A variable stores the actual value of that type.

Non-primitive types are **Reference** types

- String, Array, Class Objects, …
- A variable stores only the reference (memory address location) to point to the actual object.

PRNG Applications

Monte Carlo Simulation

Homework: 649 Lotto Lottery!

- A lottery **ticket** is **$4**.
- A ticket having **6** different numbers (from **1 to 49**) (Can be repeated)
- On a Saturday, they draw the lottery, and the winning numbers are:

  11, 43, 24, 30, 60, 43

- Match at position:
  - Match **Three** numbers to get a small prize. (**$100**)
  - Matching **Four** numbers gets a bigger prize. (**$1000**)
  - Matching **Five** is even bigger. (**$5000**)
  - Matching **ALL SIX** of the numbers you might win millions. (**$5 million** in cash)
- In the example, we got matches at position 1, 3, 4, 6 (4 numbers) = $1000
- **[Math Question]** What is the possibility of you winning by matching all 6 numbers?
- **Homework Task:**
  - Write a simple program allows you to buy a ticket with six random numbers, and generate the winning numbers and return what kind of prize you won (one game).
  - **Bonus**: imagine you buy up to 100,000 tickets, can you figure out if you actually profit or loss in a long run?

athsisfun.com/data/lottery.html

Machine Learning & Deep Learning

Homework: 649 Lotto Lottery!

- A lottery **ticket** is **$4**.
- A ticket having **6** different numbers (from **1 to 49**) (Can be repeated)
- On a Saturday, they draw the lottery, and the winning numbers are:

11, 43, 24, 30, 60, 43

- Match at each position:
  - Match **One or Two** numbers to get a small prize. (**$10**)
  - Match **Three** numbers to get a small prize. (**$100**)
  - Matching **Four** numbers gets a bigger prize. (**$1000**)
  - Matching **Five** is even bigger. (**$5000**)
  - Matching **ALL SIX** of the numbers you might win millions. (**$5 million** in cash)
- In the example, we got matches at position 1, 3, 4, 6 (4 numbers) = $1000
- **[Math Question]** What is the possibility of you winning by matching all 6 numbers?
- **Homework Task:**
  - Write a simple program allows you to buy a ticket with six random numbers, and generate the winning numbers and return what kind of prize you won (one game).
  - **Bonus**: imagine you buy up to 100,000 tickets, can you figure out if you actually profit or loss in a long run?

mathsisfun.com/data/lottery.html

Lecture 4

Control Flow Statement & Overloading

Lecture 4 - Control Flow Statement & Overloading

- **Casting** types.
- Control flow statements:
  - Decision making statements:
    - **If**
    - **If...else**
    - **Switch**
  - Loop statements:
    - **While**
    - **Do while**
    - **For**
  - Jump statements:
    - **Break** statement
    - **Continue** statement
- **Static** Keyword
  - Static Method
  - Static Variable
- **Overloading**
  - Method Overloading
  - Constructor Overloading
- **Final** Keyword
  - Constant Variable

Casting among types

- Convert one data type to another data type.

(int) 3.2 casts **double** value 3.2 to an **int**

any number type

any number expression

narrow — may be automatic cast → wider

byte **short** **int** **long** **float** **double**

← must be explicit cast, may truncate

char is a number type: **(int) 'V'** **(char) 86**

Unicode repr. in decimal: 86

'V'

Unicode: 16-bit char repr. Encodes chars in just about all languages. In java, use hexadecimal (base 16) char literals:

[Question]: Which example has automatic cast, and which one has explicit cast?

```java
int myInt = 9;
double myDouble = myInt;
System.out.println(myInt);
```

```java
double myDouble = 9.4;
int myInt = (int) myDouble;
System.out.println(myInt);
```

If Statement

- **Three** types of decision making statements:
  - **if** statement:
    - Performs an action, if a condition is true; skips it, if false.
    - **Single-selection statement**—selects or ignores a single action (or group of actions).
  - **if...else** statement:
    - Performs an action if a condition is true and performs a different action if the condition is false.
    - **Double-selection statement**—selects between two different actions (or groups of actions).
  - **switch** statement
    - Performs one of several actions, based on the value of an expression.
    - **Multiple-selection statement**—selects among many different actions (or groups of actions).

Control & Loop Flow Diagram

Here is an UML of Lottery Game Example

Lecture 5

Array & ArrayList

Lecture 5 - Array & ArrayList

- **Increment and Decrement Operators**
- **Scope of Declarations**
  - this keyword
- **Array:**
  - Declare and Create Array
  - Loop through Array
  - Pass Arrays to Methods
    - Pass by Value vs Pass by Reference
  - Multidimensional Arrays
  - Class Arrays for helper methods
- **ArrayList** (Collections Class)
- **Array vs ArrayList**

- For example, the three assignment statements:

```
passes = passes + 1;
failures = failures + 1;
studentCounter = studentCounter + 1;
```

- can be written more concisely with compound assignment operators as

```
passes += 1;
failures += 1;
studentCounter += 1;
```

- prefix increment operators as

```
++passes;
++failures;
++studentCounter;
```

- postfix increment operators as

```
passes++;
failures++;
studentCounter++;
```

Meme about Scope of Declaration

Array

- An array is **a group of variables** (called elements or components) containing **values of the same type**.
- Arrays are **objects**, so they're considered **reference types**.

```
int[ ] x = new int[5];
x[0] = 34;
x[1] = 21;
x[2] = 2;
x[3] = 66;
x[4] = 567;
```

The Nature of Multidimensional Array

- Two dimensional array is one type of multidimensional array.

1-Dimensional Array

2-Dimensional Array

3-Dimensional Array

ArrayList & Array Example

```java
int[] arr = new int[3];
arr[0] = 1;
arr[1] = 2;
arr[2] = 3;
// We cannot add more elements to array arr[]

// ArrayList
// Need not to specify size
ArrayList<Integer> arrL = new ArrayList<Integer>();

// Adding elements to ArrayList object
arrL.add(1);
arrL.add(2);
arrL.add(3);
arrL.add(4);
// We can add more elements to arrL

// Print and display Arraylist elements
System.out.println(arrL);
// Print and display array elements
System.out.println(Arrays.toString(arr));
```

Output:

```
[1, 2, 3, 4]
[1, 2, 3]
```

ArrayList

Array

What do you do when you get stuck?

Lecture 6

Inheritance

Lecture 6 - Inheritance

- Inheritance:
  - Definition and Examples
  - Types of Inheritance
  - UML Diagram
  - Animal Inheritance Example
    - Without Inheritance
    - With Inheritance
    - Method Overriding
  - Constructors in Subclasses
    - Keyword Super
  - Method Overriding
    - Keyword Super
  - Access Modifier
    - Protected
  - Exercise for University Staff & Lecturer

Inheritance

- Inheritance is a way to **reuse classes by expanding them into more specific types of classes.**
- Inheritance allows a **child class (subclass)** to **inherit the attributes and the methods** of a **parent class (superclass).** So **a child class can do anything that the parent class can do**!

- **A child class**
  - can have **its own attributes and methods.**
  - A child class can customize methods that it inherits from its parent class (**method overriding**)

- Inheritance represents **IS-A relationship** between parent and child objects.

- Inheritance promotes the idea of **code reuse** to **reduces code repetition** as classes can share similar common logic, structure, attributes and methods.

Examples of Superclass, Subclasses

- **Superclass** tend to be “more general” and **Subclass** is more “more specific.”

Parent Class (Superclass)	Child Class (Subclasses)
Vehicles	Car, Truck, Boat, Bicycle
Shape	Circle, Triangle, Rectangle, Cube
UniversityStaff	Lecturer, TeachingAssistant
Animal	Dog, Cat, Spider, Duck

Which access modifier to use?

- For beginners, keep it simple:
  - Class: **public**
  - Attributes/Instance Variables: **private**
  - Methods (including getters and setters): **public**
  - Class Variables (static keyword): **public**
  - Class Methods (static keyword): **public**
  - Constants (final keyword):
    - Class Constant (with static keyword): **public**
    - Normal Constant (without static keyword): **private**

- **The short answer is, you should make everything as restricted as possible** (private over protected, and protected over public) **while still allowing things to get done.**
- When it comes to deciding between public and private, it all comes down to how widely used you expect the class to be. If you think a class is reusable across many projects, then you might as well make it public from the beginning.

Lecture 7

Polymorphism

Lecture 7 - Polymorphism

- **Polymorphism**
  - Method overriding in Inheritance
    - Zoo Example
  - **Abstraction**
    - Abstract Class
    - Abstract Method
    - Examples:
      - Zoo Example
      - Company Payroll Example
  - **Interface**
    - Interface in real life examples
    - Upgrade Company Payroll with Invoices Example
  - Abstract vs Interface vs Composition

Polymorphism

- Polymorphism literally means **“many shapes”.**
- The **same method** is called in **different types of objects** has **different results**.
- Enables you to **“program in the general”** rather than **“program in the specific.”**
- Polymorphism **enables you to write programs that process objects of subclasses that share the same superclass as if they’re all objects of the superclass**; this **can simplify programming.**

Abstraction in Shape

The class Shape does not provide any implementation for draw() ; basically there is no code, and this is what makes the method abstract (providing any code would make the method concrete).

```
public abstract class Shape {

    public abstract void draw(); // no implementation

}
```

We want the subclasses to provide the implementation. Let's look at the Circle and Rectangle classes:

```
public class Circle extends Shape {

    public void Draw() {System.out.println ("Draw a Circle")};

}

public class Rectangle extends Shape {

    public void Draw() {System.out.println ("Draw a Rectangle")};

}
```

```
circle.draw();          // draws a circle
rectangle.draw();      // draws a rectangle
```

The Draw method can be invoked for every single shape in the system, and invoking each shape produces a different result:
- Invoking the Draw method on a Circle object draws a circle.
- Invoking the Draw method on a Rectangle object draws a rectangle.

In essence, sending a message to an object evokes a different response, depending on the object. This is **the essence of polymorphism.**

Abstract VS Interface vs Composition?

- The obvious question is this: **If an abstract class can provide the same functionality as an interface, why do Java bother to provide an interface**?
- If both **abstract classes and interfaces provide abstract methods, what is the real difference between the two?**
- As we saw before, **an abstract class provides both abstract and concrete methods, whereas an interface provides only abstract methods. Why is there such a difference?**

- In a nutshell, Java **build objects in three ways**: **inheritance**, **interfaces**, and **composition**.
- This example illustrates when you should use which one. When do you choose an abstract class? When do you choose an interface? When do you choose composition?
- You should be familiar with the following concepts:
  - Dog is a Mammal , so the relationship is **inheritance**.
  - Dog implements Nameable , so the relationship is an **interface**.
  - Dog has a Head , so the relationship is **composition**.

Although **inheritance** is a **strict is-a relationship**, an **interface** is **not** quite. For example:

- A dog is a mammal.
- A toy is not a mammal.

Thus, a Toy class could not inherit from the Mammal class.
However, an interface applies for the various classes. For example:

- A dog is nameable.
- A teddy bear is nameable.

```java
public abstract class Mammal {

    public void generateHeat() {System.out.println("Generate heat");}

    public abstract void makeNoise();

}
```

```java
public interface Nameable {

    String getName();
    void setName (String aName);

}
```

```java
public class Head {

    String size;

    public String getSize() {

        return size;

    }

    public void setSize(String aSize) { size = aSize;}

}
```

```java
public class Dog extends Mammal implements Nameable {

    String name;

    Head head;

    public void makeNoise(){System.out.println("Bark");}

    public void setName (String aName) {name = aName;}
    public String getName () {return (name);}

}
```

Lecture 8

Graphic User Interface (GUI)

Lecture 8 - Graphic User Interface (GUI)

- **Mid-term Exam**
  - Possible Topics
  - Format
- **GUI** in Java
  - Swing Components
  - Swing vs JavaFX vs AWT
  - JOption-Panel (DialogBox
  - Java 2D API
    - Java Coordinate System
    - Draw Lines
    - Draw Rectangles and Ovals
    - Draw Colors
    - Draw Polygons and Polylines
  - Text Fields, Buttons and Menu Nagivations.
  - Event Handler (Mouse & Keyboard)
  - Layout:
    - BorderLayout Frame & GridLayoutFrame

Sample GUI of Java's Swing GUI APIs

SwingSet3 application demonstrates many of Java's Swing GUI components.

https://mlapshin.com/index.php/projects/swingset3/

Color in Computer

Lecture 9

Generic Collections

Lecture 9 - Generic Collections

- **Mid-term Exam Result**
  - Score Distribution
  - Statistics
  - Solutions
- **Java Generic Collections**
  - Type-Wrapper Classes for Primitive Types
    - Autoboxing vs Auto-unboxing
  - List
    - ArrayList
    - Vector
    - LinkedList
  - Sets
    - HashSet
    - TreeSet
  - Maps
    - Hashtable
    - HashMap
    - TreeMaps
- HackerRank
  - Introduction
  - Coding Challenge

Collections

- Java **Collections** framework:
  - **Prebuilt data structures.**
  - **A data structure is a particular way of organizing data so that it can be used effectively.**
  - Interfaces and Methods for manipulating those data structures.
  - A collection is a data structure—actually, an object—that can hold references to other objects.
  - Usually, collections contain references to objects that are all of the same type.

Interface	Description
`Collection`	The root interface in the collections hierarchy from which interfaces `Set`, `Queue` and `List` are derived.
`Set`	A collection that does not contain duplicates.
`List`	An ordered collection that can contain duplicate elements.
`Map`	A collection that associates keys to values and cannot contain duplicate keys.
`Queue`	Typically a first-in, first-out collection that models a waiting line; other orders can be specified.

Fig. 20.1 | Some collections-framework interfaces.

Challenge Time!
Real Job Interview Question: Ransom Note

https://www.hackerrank.com/challenges/ctci-ransom-note/problem

Hint: you should use HashMap.

Lecture 10

Generic Class

Lecture 10 - Generic Class

- Generic
  - Generic Class
    - Bounded type parameters
  - Generic Method

Type Parameter

```
class Data<T>{
    T data;
    public Data(T data) { this.data = data; }
    public T getData() {return data; }
}

public class Main {
    public static void main(String[] args) {
        Data<Integer> a = new Data<>(1);
    }
}
```

Type Argument

Parameterized Type

The Problem

Which data type do you want?
Integer
Double
String
imgflip.com

To create object/instance of Generic class

- Create a new generic class with a type placeholder T:

```
public class ClassName <T>{
…
}
```

- Use that generic class to create an object from it and specific what datatype will be replace T.

```
ClassName <Type> objectName = new ClassName <Type>();
```

Simple Generic Class Example

Type Parameter

```
class Data<T>{
    T data;
    public Data(T data) { this.data = data; }
    public T getData() {return data; }
}

public class Main {
    public static void main(String[] args) {
        Data<Integer> a = new Data<>(1);
    }
}
```

Type Argument

Parameterized Type

Problem

```java
public class MyNumberClass <T>{
        T myNumber;

        public MyNumberClass(T myNumber) {
            this.myNumber = myNumber;
        }

        public double square(){
            return myNumber * myNumber;
        }
}
```

Operator '*' cannot be applied to 'T', 'T'

```java
public class Main {
    public static void main(String[] args) {
        MyNumberClass<Integer> myNumber = new MyNumberClass<>(4);
        System.out.println(myNumber.square());
    }
}
```

Solve it with Bound Type Parameters

```java
public class MyNumberClass <T extends Number>{
        T myNumber;

        public MyNumberClass(T myNumber) {
            this.myNumber = myNumber;
        }

        public double square(){
            return myNumber.intValue() * myNumber.doubleValue();
        }
}
```

```java
public class Main {
    public static void main(String[] args) {
        MyNumberClass<Integer> myNumber = new MyNumberClass<>(4);
        System.out.println(myNumber.square());
    }
}
```

Output:

```
16.0
```

Lecture 11

OO Design Principles (SOLID)

Lecture 11 - OO Design Principles (SOLID)

- Object Oriented Design Principles: SOLID
  - S: Single responsibility
  - O: Open/closed principle
  - L: Liskov substitution principle
  - I: Interface segregation principle
  - D: Dependency inversion principle

SOLID Summary

Principles for maintainable object-oriented code

Single Responsibility Principle

Each class has a single purpose. All its methods should relate to function.
Reasoning: Each responsibility could be a reason to change a class in the future. Fewer responsibilities → fewer opportunities to introduce bugs during changes.
Example: Split formatting & calculating of a report into different classes.

Open / Closed Principle

Classes (or methods) should be open for extension and closed for modification. Once written they should only be touched to fix errors. New functionality should go into new classes that are derived. This is popularly interpreted to advocate inheriting from an abstract base class.
Reasoning: Again you lower the odds of breaking existing code.

Liskov Substitution Principle

You should be able to replace an object with any of its derived classes. Your code should never have to check which sub-type it's dealing with.
Reasoning: Prevents awkward type checking and weird side-effects.

Interface Segregation Principle

Define subsets of functionality as interfaces.
Reasoning: Small, specific interfaces lead to a more decoupled system than a big general-purpose one.
Example: A `PersistenceManager` implements `DBReader` & `DBWriter`.

Dependency Inversion Principle

High level modules should not depend on low-level modules. Instead, both should depend on abstractions. Abstractions should not depend on details. Details should depend upon abstractions
Reasoning: High-level modules become more reusable if they are ignorant of low-level module implementation details.
Examples: 1) Dependency Injection. 2) Putting high-level modules in different packages than the low-level modules it uses.

SRP: Splitting a non-SRP class

OCP: Modifying OldClass instead of extending

LSP: seamless substitution for the B class in existing code

ISP: Only inherit applicable methods

DIP: Changes to Objects A and B will not affect the other

Lecture 12

Design Pattern

Lecture 12 - Design Pattern

- Design Patterns:
  - Creational
    - Singleton
    - Factory Method
  - Structural
  - Behavioral

Factory Design Pattern

OOP Concepts vs OOP Principles vs Design Pattern

Types of Design Patterns

There are three categories of these patterns:

- **Creational Patterns**
  - Focused on the way objects are created.
- **Structural Patterns**
  - Focused on creating objects into larger structures which gives developers the advantage to keep groups of children together and also abstract objects to simplify applications.
- **Behavioral Patterns**
  - Focused on combining algorithms and responsibilities together which concerned with communication between objects.

Creational Design Pattern

These design patterns focused on the way objects are created:

- **Abstract Factory (⭐⭐⭐)**: Lets you produce families of related objects without specifying their concrete classes.

- **Builder(⭐⭐⭐)**: Lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.

- **Factory Method(⭐⭐⭐)**: Provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

- **Prototype(⭐⭐)**: Lets you copy existing objects without making your code dependent on their classes.

- **Singleton(⭐⭐)**: Lets you ensure that a class has only one instance, while providing a global access point to this instance.

Structural Design Pattern

These design patterns focused on creating objects into larger structures:

- **Adapter(⭐⭐⭐)**: Allows objects with incompatible interfaces to collaborate.

- **Bridge(⭐)**: Lets you split a large class or a set of closely related classes into two separate hierarchies—abstraction and implementation—which can be developed independently of each other.

- **Composite(⭐⭐)**: Lets you compose objects into tree structures and then work with these structures as if they were individual objects.

- **Decorator(⭐⭐)**: Lets you attach new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors.

- **Facade(⭐⭐)**: Provides a simplified interface to a library, a framework, or any other complex set of classes.

- **Flyweight(⭐)**: Lets you fit more objects into the available amount of RAM by sharing common parts of state between multiple objects instead of keeping all of the data in each object.

- **Proxy(⭐)**: Lets you provide a substitute or placeholder for another object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.

Behavioral Design Pattern

These design patterns focused on the communication between objects.:

- **Chain of responsibility(⭐):** Lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

- **Command(⭐⭐⭐)**: Turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method arguments, delay or queue a request's execution, and support undoable operations.

- **Iterator(⭐⭐⭐)**: Lets you traverse elements of a collection without exposing its underlying representation (list, stack, tree, etc.).

- **Mediator(⭐⭐)**: Lets you reduce chaotic dependencies between objects. The pattern restricts direct communications between the objects and forces them to collaborate only via a mediator object.

- **Memento(⭐)**: Lets you save and restore the previous state of an object without revealing the details of its implementation.

- **Observer(⭐⭐⭐)**: Lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.

- **State(⭐⭐)**: Lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.

- **Strategy(⭐⭐⭐)**: Lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.

- **Template method(⭐⭐):** Defines the skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure.

- **Visitor(⭐)**: Lets you separate algorithms from the objects on which they operate.

Singleton: What?

- Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.
- Singleton is a creational design pattern that lets you **ensure that a class has only one instance**, while **providing a global access point to this instance**.

Factory Method: What?

- Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

Lecture 13

Web Scraping

Lecture 13 - Web Scraping

- **What** is Web Scraping?
- **Why** is it important?
- **How** to use it safely and legally?
- HTML, CSS
- Introduction to **JSoup** Library
- JSoup workflow
- VNExpress Scrapping Demo
- Manga Scrapping Demo
- Potential issues with Web Scraping

	title	url	image_url	description
0	Đại sứ Mỹ: Chiến lược với châu Á sẽ không đổi ...	https://vnexpress.net/dai-su-my-chien-luoc-voi...	https://vcdn1-vnexpress.vnecdn.net/2020/09/04/...	Đại sứ Mỹ tại Việt Nam Daniel Kritenbrink cho ...
1	Phương án sửa Bảo tàng Mỹ thuật TP HCM	https://vnexpress.net/phuong-an-sua-bao-tang-m...	https://vcdn1-vnexpress.vnecdn.net/2020/09/24/...	Chủ đầu tư cao ốc từ giác Bến Thành gửi Sở Văn...
2	Việt Nam mong muốn phát huy quan hệ hợp tác to...	https://vnexpress.net/viet-nam-mong-muon-phat-...	https://vcdn1-vnexpress.vnecdn.net/2020/09/25/...	Tổng Bí thư, Chủ tịch nước Nguyễn Phú Trọng ch...
3	Bỗng dưng mắc nợ ngân hàng dù không vay vốn	https://vnexpress.net/bong-dung-mac-no-ngan-ha...		Nhiều khách hàng bỗng dưng phát hiện mắc nợ cô...
4	Cô dâu Việt 'đổi đời' nhờ mẹ chồng Tây	https://vnexpress.net/co-dau-viet-doi-doi-nho-...	https://vcdn1-giadinh.vnecdn.net/2020/09/25/11...	AnhĐang tuyệt vọng vì bạn trai cắt đứt mọi liê...
5	Kịch bản Trump từ chối rời Nhà Trắng gây lo ngại	https://vnexpress.net/kich-ban-trump-tu-choi-r...	https://vcdn1-vnexpress.vnecdn.net/2020/09/24/...	Trump mới đây cho biết ông không hứa rời Nhà T...
6	Giá xăng ngày mai có thể giảm tiếp	https://vnexpress.net/gia-xang-ngay-mai-co-the...	https://vcdn1-kinhdoanh.vnecdn.net/2020/09/25/...	Thị trường thế giới đi xuống so với chu kỳ trư...
7	Cháy khách sạn, một người chết	https://vnexpress.net/chay-khach-san-mot-nguoi...	https://vcdn1-vnexpress.vnecdn.net/2020/09/25/...	TP HCMTiến vào khu vực lửa đang bùng lên ở tần...
8	Chuẩn bị khởi công 3 dự án cao tốc Bắc Nam	https://vnexpress.net/chuan-bi-khoi-cong-3-du-...	https://vcdn1-vnexpress.vnecdn.net/2020/09/25/...	Các dự án Mai Sơn - quốc lộ 45, Vĩnh Hảo - Pha...
9	Nga - Trung công kích Mỹ tại Liên Hợp Quốc	https://vnexpress.net/nga-trung-cong-kich-my-t...	https://vcdn1-vnexpress.vnecdn.net/2020/09/25/...	Đại sứ Trung Quốc tại Liên Hợp Quốc cáo buộc M...
10	Bayern đoạt Siêu Cup châu Âu	https://vnexpress.net/bayern-doat-sieu-cup-cha...	https://vcdn1-thethao.vnecdn.net/2020/09/25/ba...	HungaryThắng ngược Sevilla 2-1 ở hiệp phụ tối ...
11	Nhà phân phối mới xe Nissan tại Việt Nam là ai?	https://vnexpress.net/nha-phan-phoi-moi-xe-nis...	https://vcdn1-vnexpress.vnecdn.net/2020/09/25/...	Công ty TNHH Phát triển Công nghiệp Ôtô Việt N...
12	Nạn nhân bị cây đè từ vong sau mổ	https://vnexpress.net/nan-nhan-bi-cay-de-tu-vo...	https://vcdn1-suckhoe.vnecdn.net/2020/09/25/ca...	TP HCMNạn nhân 35 tuổi, bị cây đổ trong cơn mư...
13	300 năm người Thái ấp ủ tham vọng 'kênh đào Pa...	https://vnexpress.net/300-nam-nguoi-thai-ap-u-...		Cứ vài năm, ý tưởng đào con kênh qua Eo đất Kr...
14	Loạt dịch vụ Google gặp sự cố	https://vnexpress.net/loat-dich-vu-google-gap-...	https://vcdn1-sohoa.vnecdn.net/2020/09/25/goog...	Gmail, Drive, YouTube và nhiều dịch vụ Google ...
15	22 ngày không lây nhiễm nCoV cộng đồng	https://vnexpress.net/22-ngay-khong-lay-nhiem-...		Bộ Y tế sáng 25/9 không ghi nhận thêm ca dương...

What is Web Scraping?

Web Pages

Web Scraping

Structured Data

Web Component Overview

VNExpress Homepage Scraping Result

```
Title: Đề xuất gần 113 dự án dùng vốn từ gói phục hồi kinh tế
Article: https://vnexpress.net/de-xuat-gan-113-du-an-dung-von-tu-goi-phuc-hoi-kinh-te-4466119.html
Description: Số dự án, nhiệm vụ dùng vốn từ gói phục hồi kinh tế được đề xuất còn 113, giảm khoảng một phần ba so với con số đưa ra cách đây hơn một tháng.
------------------------------------
Title: Đỏ đen tuổi 18
Article: https://vnexpress.net/do-den-tuoi-18-4465661.html
Description: Tôi không phản đối casino nhưng bây giờ chưa phải lúc để người Việt 18 tuổi vào chơi.
------------------------------------
Title: Sóng ngầm cắt lỗ căn hộ của nhà đầu tư F0
Article: https://vnexpress.net/song-ngam-cat-lo-can-ho-cua-nha-dau-tu-f0-4465898.html
Description: TP HCMVài tháng nay, anh Điền rao bán lỗ 100 triệu đồng căn hộ một phòng ngủ, giá 1,7 tỷ đồng đang xây móng, song chẳng ai hỏi mua.
------------------------------------
Title: Nga tuyên bố sắp kiểm soát toàn bộ tỉnh miền đông Ukraine
Article: https://vnexpress.net/nga-tuyen-bo-sap-kiem-soat-toan-bo-tinh-mien-dong-ukraine-4466155.html
Description: Bộ trưởng Shoigu thông báo lực lượng Nga và phe ly khai sắp kiểm soát toàn bộ tỉnh Lugansk ở miền đông Ukraine sau gần ba tháng chiến sự.
------------------------------------
Missing Article Title!
Article: /the-thao/sea-games-31?utm_source=VnExpress&utm_medium=Box&utm_campaign=SeaGames31_PC&utm_content=SEAGames31
Missing Article Description!
------------------------------------
Title: Nghệ sĩ Phượng Loan gặp tai nạn giao thông
Article: https://vnexpress.net/nghe-si-phuong-loan-gap-tai-nan-giao-thong-4466095.html
Description: Nghệ sĩ cải lương Phượng Loan ngã từ trên xe máy khi đi hát đám cưới, được chẩn đoán xuất huyết não, gãy xương má...
```

Italian Recipe Scrapping Homework

https://foodnetwork.co.uk/italian-family-dinners/

Scrap all Italian recipes, which should have:

- Recipe title
- Short description
- Preparation time
- Cook time
- Number of serves

Our Challenge

- Let’s scrap one of the best manga websites, “MangaDoom”!
  https://www.mngdoom.com/

Let's pick a best manga to scrap them!

- "One Piece" is my favourite manga of all time! Let's scrap the whole manga!

I'm a collector myself! Web Scraping is awesome!

Ultimate Challenge

- **MangaSee** is **the only website** which got the **FULL-COLORED** chapters of big mangas like One Piece and Bleach!!!
- [Task] Can you figure out how to scrap either full-colored One Piece or Bleach?
  - https://mangasee123.com/manga/One-Piece-Digital-Colored-Comics
  - https://mangasee123.com/manga/Bleach-Color

One of The Best Manga Downloader Project

- https://github.com/riderkick/FMD
- The Free Manga Downloader is a free open source application written in Object Pascal for managing and downloading manga from various websites.
- Based on what you have learned so far about web scraping, now you can actually start to contribute to this open source project to help the Manga community!

Lecture 14

Final Lecture - Revision

Lecture 14 - Final Lecture - Today Revision

- Course revision:
  - Go through topics we have learnt for each week
- Final exam format:
  - Number of questions
  - Topics
  - Time

Final Exam Format

- Offline
- Open book (Lecture notes, books, … are allowed BUT NO mobile phone, internet, google or laptop)
- 4 Questions, with topics could be related to:
  - OO Concepts (keywords, ideas, …)
  - Generic (Collections, Class)
  - OO Design Principles (SOLID)
  - Design Patterns
- 120 minutes

Final Game Project

- Your team present your project will be on the dates:
  - For Mr. Nhân's Lab: Friday (10/06)
  - For Mr. Tom's Lab: Saturday (11/06)
- Time for your presentation: 12 Minutes
  - Could be break down into:
    - 9 minutes presenting slides
    - 3 minutes of live demo your game
- The lab instructor will ask some questions for 3-5 minutes after the presentation finishes.
- Marking Criteria for the final Project:
  - 20% report (README file)
  - 30% presentation
  - 50% demo
- [Link to register for your lab dates!]
- [Link for the requirement document!]

imgflip.com

JAKE-CLARK.TUMBLR

Final Thought & Message

The Secret of Success: Get 1% Better Every Day

- The only idea is to improve slightly everyday! Learn something new everyday!
- Let's prove this point:

The Power of Tiny Gains

1% better every day $1.01^{365} = 37.78$

1% worse every day $0.99^{365} = 0.03$

THE PLATEAU OF LATENT POTENTIAL

FIGURE 2: We often expect progress to be linear. At the very least, we hope it will come quickly. In reality, the results of our efforts are often delayed. It is not until months or years later that we realize the true value of the previous work we have done. This can result in a "valley of disappointment" where people feel discouraged after putting in weeks or months of hard work without experiencing any results. However, this work was not wasted. It was simply being stored. It is not until much later that the full value of previous efforts is revealed.

What's next for you?

What's next for Tom?

```
Career tomCareer = new Career("PhD");
while (True) {
    tomCareer.study();
    tomCareer.teach();
    tomCareer.work();
}
```

Thank you for listening!

“Always be learning!”

Tom Huynh