Encapsulation

Lab 4

TA : Hyuna Seo, Kichang Yang, Minkyung Jeong, Jaeyong Kim

SEOUL NATIONAL UNIVERSITY

Announcement

- You should finish the lab practice and submit your job to eTL before the next lab class starts (Wednesday, 7:00 PM).
- The answer of the practice will be uploaded after the due.

Overview

- Lecture Recap
  - Access Modifier
  - Getter and Setter
  - Java Packages
- **Tip :** Import Built-in Packages
- **Tip :** Java API Documents
- Problem 1. Implement Dice class
- Problem 2. RockPaperScissors class
- Problem 3. Implement Platform class

Recap: Encapsulation in Java

- Encapsulation is implemented using **classes, access modifiers,** and **setters and getters.**
- Access modifiers restrict access to attributes at different levels.
  - private, public, protected, default
- Setters and getters allow controlled data access.
  - Prevent unexpected changes from other objects.

Recap: Access Modifier `private`

Class Definition

```
class Pizza {
   private String topping = “Pineapple”;
}
```

Not accessible by other classes

main Method

```
Pizza hawaiian = new Pizza();
System.out.println(hawaiian.topping);
```

Output

```
java: topping has private access in Pizza
```

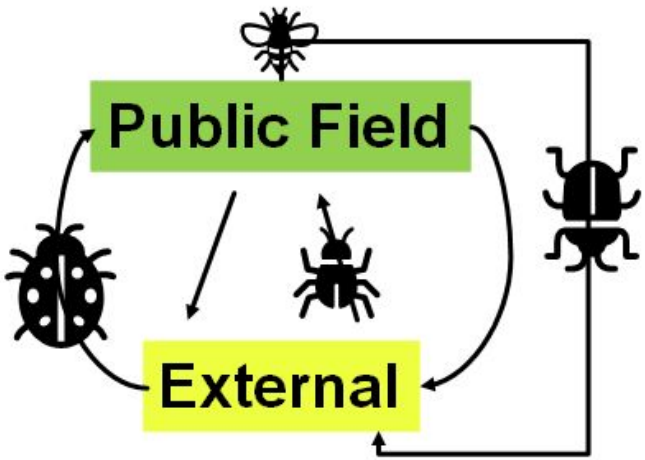
Recap: Getter and Setter

- Public methods to access or modify **private** attributes.
- Getters and setters are not the Java syntax, but a convention to implement encapsulation.

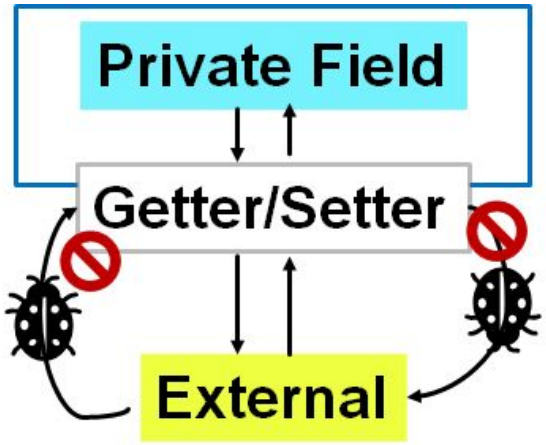
```
private int income;
public int getIncome() {
    // Getter format : public type getXXX()
    return income;
}
public void setIncome(int income) {
    // Setter format : public void setXXX (type xxx)
    this.income = income;
}
```

Recap: Role of Getter and Setter

- The access to private attributes can be done only through getters and setters.



Direct Access

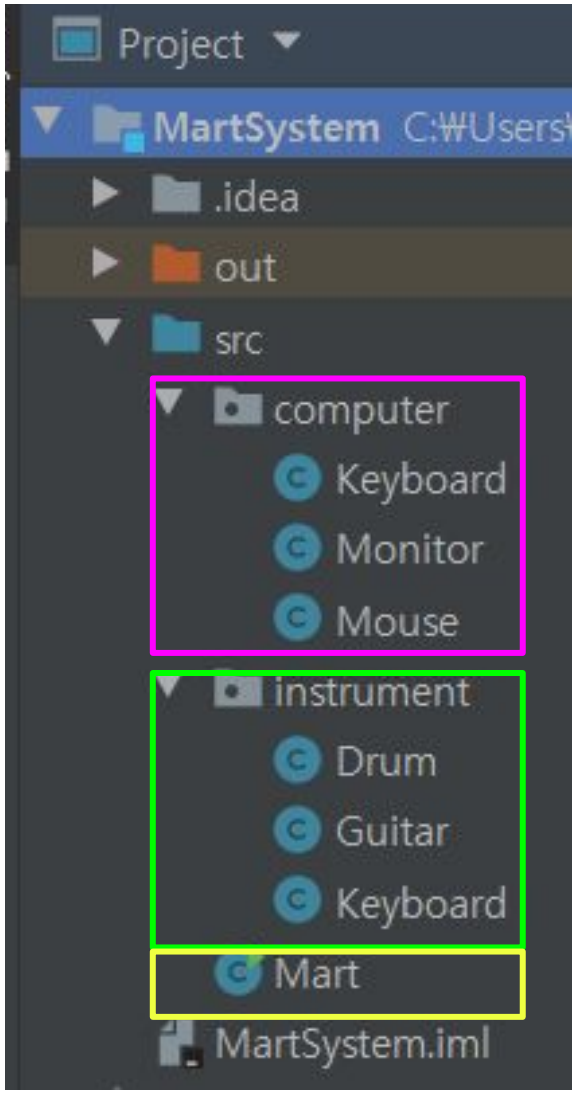


Indirect Access

Recap:
Access Validation with Getter/Setter

```
class Book {
    private String title;
    public void setTitle(String title) {
        if (title == null || title.equals("")) {
            System.out.println(
                    "Title cannot be null or empty");
        } else {
            this.title = title;
        }
    }
    public void getTitle() {
        return title;
    }
}
```

Recap: Package Structure Example



Imagine a case with 3 packages: Computer, Instrument, and Default.

Computer package has 3 classes.
(Keyboard, Monitor, Mouse)

Instrument package has 3 classes.
(Drum, Guitar, Keyboard)

Default package has 1 class.
(Mart : this class has main() function.)

Recap: Putting a Class in a Package

Class Definition

```java
package computer;

public class Keyboard {
    private int id;
    private String name;

    public Keyboard(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public void printCompanyInfo() {
        if (this.id % 10 == 1) {
            System.out.println(name + ": Corsair's device");
        } else if (this.id % 10 == 2) {
            System.out.println(name + ": Razor's device");
        } else {
            System.out.println(name + ": Realforce's device");
        }
    }
}
```

Recap: Putting a Class in a Package

Class Definition

```java
package instrument;

public class Keyboard {
   private int id;
   private String name;

   public Keyboard(int id, String name) {
       this.id = id;
       this.name = name;
   }

   public void printCompanyInfo() {
       if (this.id % 10 == 1) {
           System.out.println(name + ": Samick's device");
       } else if (this.id % 10 == 2) {
           System.out.println(name + ": Yamaha's device");
       } else {
           System.out.println(name + ": Gipson's device");
       }
   }
}
```

Recap: Using Class in a Different Package

main Method in the Mart class under the default package

```
System.out.println("Welcome to SNU Mart\n");

computer.Keyboard keyboardComputer =
        new computer.Keyboard(15523,"H Gaming keyboard");
instrument.Keyboard keyboardMusic =
        new instrument.Keyboard(131511, "Black and white
keyboard");

System.out.println("<new product of computer keyboard company
info>");
keyboardComputer.printCompanyInfo();

System.out.println("<new product of instrument keyboard company
info>");
keyboardMusic.printCompanyInfo();
```

Recap: Using Class in a Different Package

Output

```
Welcome to SNU Mart

<new product of computer keyboard company info>
H Gaming keyboard: Razor's device
<new product of instrument keyboard company info>
Black and white keyboard: Samick's device
```

Objectives

- Understand the concept package.
- Understand the necessity of APIs.

Packages

- **Package** encapsulates a group of classes.
- You can make your own package.
- There are also various useful **built-in Packages**.

Import Built-In Packages

- Use **Import Syntax** to use Built-In Packages.
- Import format for one package
  - **import + "Class path"**
  - ex) `import java.util.Scanner;`
- Import format for all classes of the package
  - **import + "Package path" + ***
  - ex) `import java.time.*;`

Java API Documents

- **API** : **A**pplication **P**rogramming **I**nterface
  - Import useful packages with rich functionality

- The detailed documentation of Java APIs

  https://docs.oracle.com/en/java/javase/11/docs/api/index.html

Why are APIs necessary?

- Because it is cumbersome to manually implement all the functionalities we need.
- So we borrow the existing ones.

Java API Documents

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Java SE 11 & JDK 11

ALL CLASSES SEARCH: Search

Java® Platform, Standard Edition & Java Development Kit Version 11 API Specification

This document is divided into two sections:

Java SE

The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

JDK

The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

All Modules | Java SE | JDK | Other Modules

1. Module List

Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing UI toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
java.instrument	Defines services that allow agents to instrument programs running on the JVM.
java.logging	Defines the Java Logging API.
java.management	Defines the Java Management Extensions (JMX) API.
java.management.rmi	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.
java.naming	Defines the Java Naming and Directory Interface (JNDI) API.
java.net.http	Defines the HTTP Client and WebSocket APIs.
java.prefs	Defines the Preferences API.
java.rmi	Defines the Remote Method Invocation (RMI) API.
java.scripting	Defines the Scripting API.
java.se	Defines the API of the Java SE Platform.
java.security.jgss	Defines the Java binding of the IETF Generic Security Services API (GSS-API).
java.security.sasl	Defines Java support for the IETF Simple Authentication and Security Layer (SASL).
java.smartcardio	Defines the Java Smart Card I/O API.
java.sql	Defines the JDBC API.
java.sql.rowset	Defines the JDBC RowSet API.

Java API Documents

Module java.base

`module java.base`

Defines the foundational APIs of the Java SE Platform.

Providers:
The JDK implementation of this module provides an implementation of the jrt file system provider to enumerate and read the class and resource files in a run-time image. The jrt file system can be created by calling `FileSystems.newFileSystem(URI.create("jrt:/"))`.

Module Graph:

java.base

Tool Guides:
java launcher, keytool

Since:
9

Packages

Exports

Package	Description
java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.lang.annotation	Provides library support for the Java programming language annotation facility.
java.lang.constant	Classes and interfaces to represent *nominal descriptors* for run-time entities such as classes or method handles, and classfile entities such as constant pool entries or `invokedynamic` call sites.
java.lang.invoke	The `java.lang.invoke` package provides low-level primitives for interacting with the Java Virtual Machine.
java.lang.module	Classes to support module descriptors and creating configurations of modules by means of resolution and service binding.
java.lang.ref	Provides reference-object classes, which support a limited degree of interaction with the garbage collector.
java.lang.reflect	Provides classes and interfaces for obtaining reflective information about classes and objects.
java.lang.runtime	The `java.lang.runtime` package provides low-level runtime support for the Java language.
java.math	Provides classes for performing arbitrary-precision integer arithmetic (`BigInteger`) and arbitrary-precision decimal arithmetic (`BigDecimal`).

2. Built-in Package List

Java API Documents

Module java.base

Package java.math

package java.math

Provides classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic (BigDecimal). BigInteger is analogous to the primitive integer types except that it pro
BigIntegers do not overflow or lose precision. In addition to standard arithmetic operations, BigInteger provides modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a fe
provides arbitrary-precision signed decimal numbers suitable for currency calculations and the like. BigDecimal gives the user complete control over rounding behavior, allowing the user to choose from a comprehe

Since:
1.1

Class Summary

Class	Description
BigDecimal	Immutable, arbitrary-precision signed decimal numbers.
BigIntege class in java.math	Immutable arbitrary-precision integers.
MathContext	Immutable objects which encapsulate the context settings which describe certain rules for numerical operators, such as those implemented by the BigDecimal class.

Enum Class Summary

Enum Class	Description
RoundingMode	Specifies a *rounding behavior* for numerical operations capable of discarding precision.

3. List of classes and enums

Java API Documents

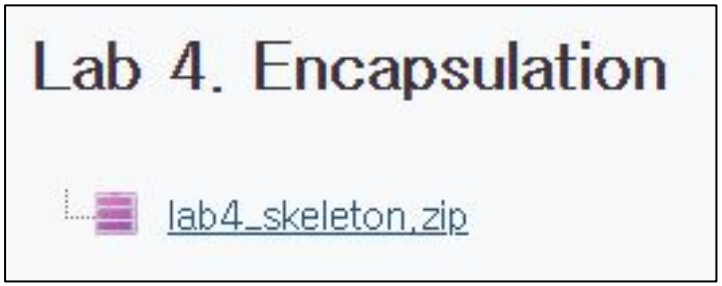
Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Met

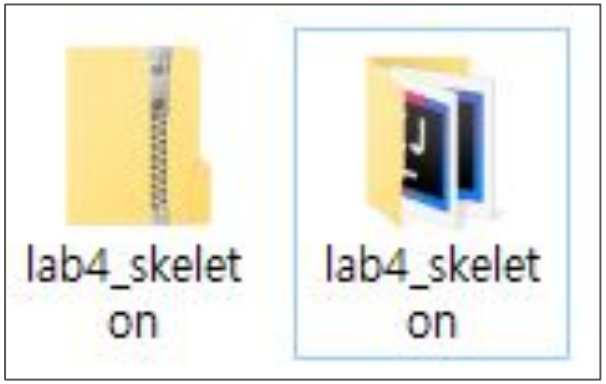
Modifier and Type	Method
BigDecimal	abs()
BigDecimal	abs(MathContext mc)
BigDecimal	add(BigDecimal augend)
BigDecimal	add(BigDecimal augend, MathContext mc)
byte	byteValueExact()
int	compareTo(BigDecimal val)
BigDecimal	divide(BigDecimal divisor)
BigDecimal	divide(BigDecimal divisor, int roundingMode)
BigDecimal	divide(BigDecimal divisor, int scale, int roundingMode)
BigDecimal	divide(BigDecimal divisor, int scale, RoundingMode roundingMode)
BigDecimal	divide(BigDecimal divisor, MathContext mc)
BigDecimal	divide(BigDecimal divisor, RoundingMode roundingMode)
BigDecimal[]	divideAndRemainder(BigDecimal divisor)
BigDecimal[]	divideAndRemainder(BigDecimal divisor, MathContext mc)

4. Members of the selected classes

Before going in to the problem..



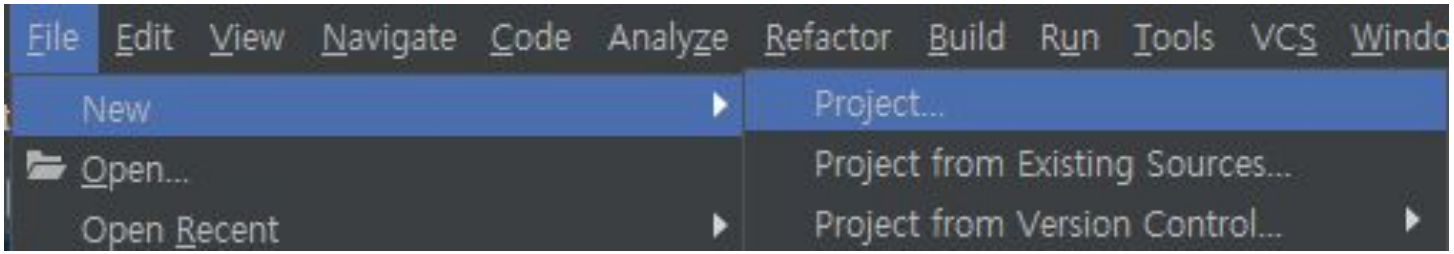
1. Download the skeleton



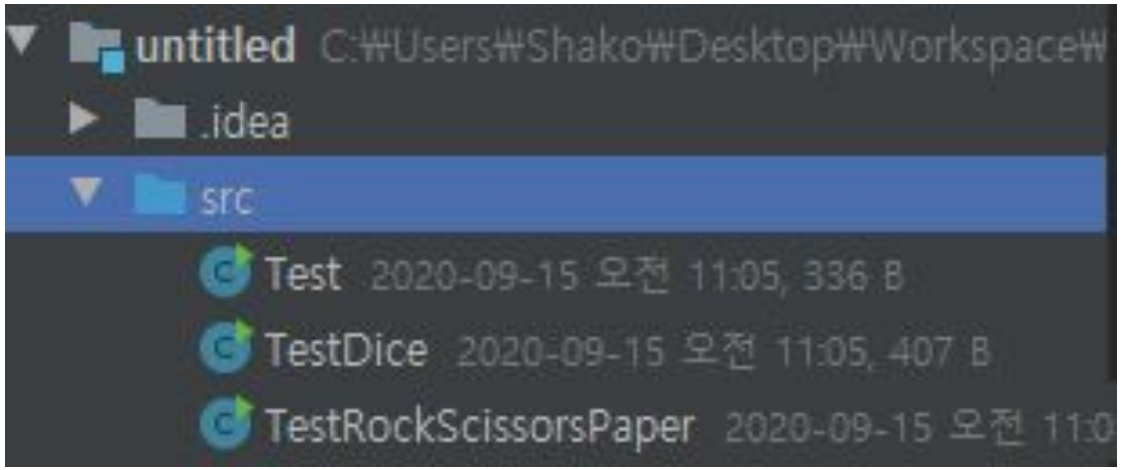
2. Extract the Zip file

3. Check the .java files and Copy them

Before going in to the problem..



4. Create a new project

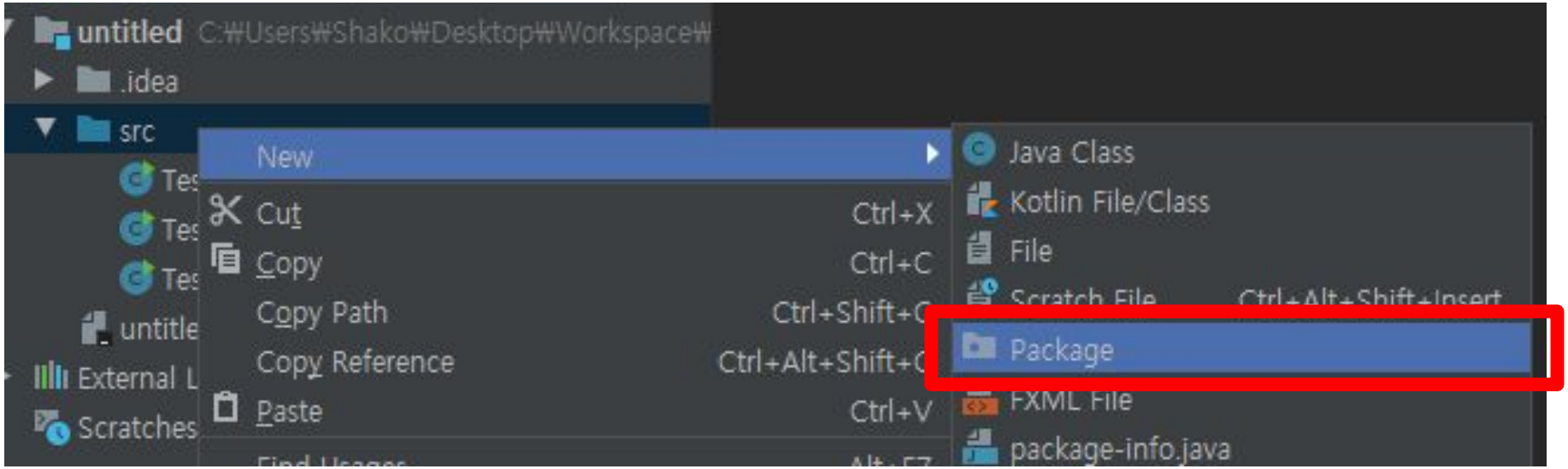


5. Paste the skeleton code classes to the src dir

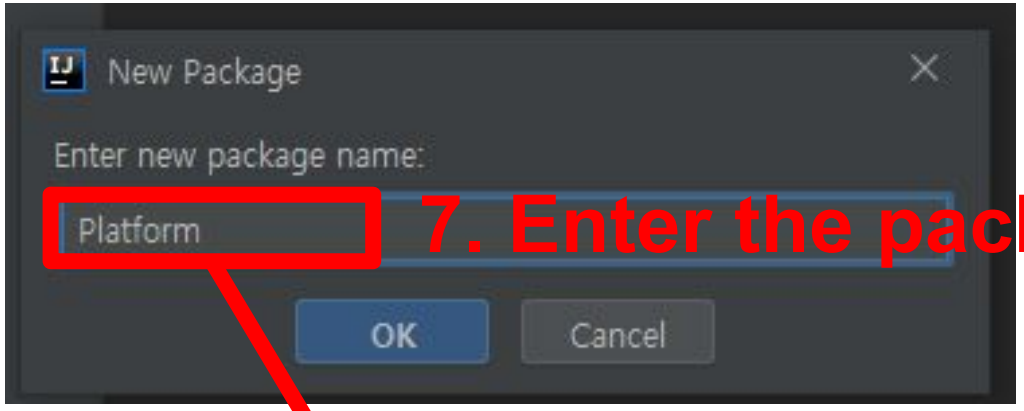
Don't mind the red line under the file name.

Create Packages

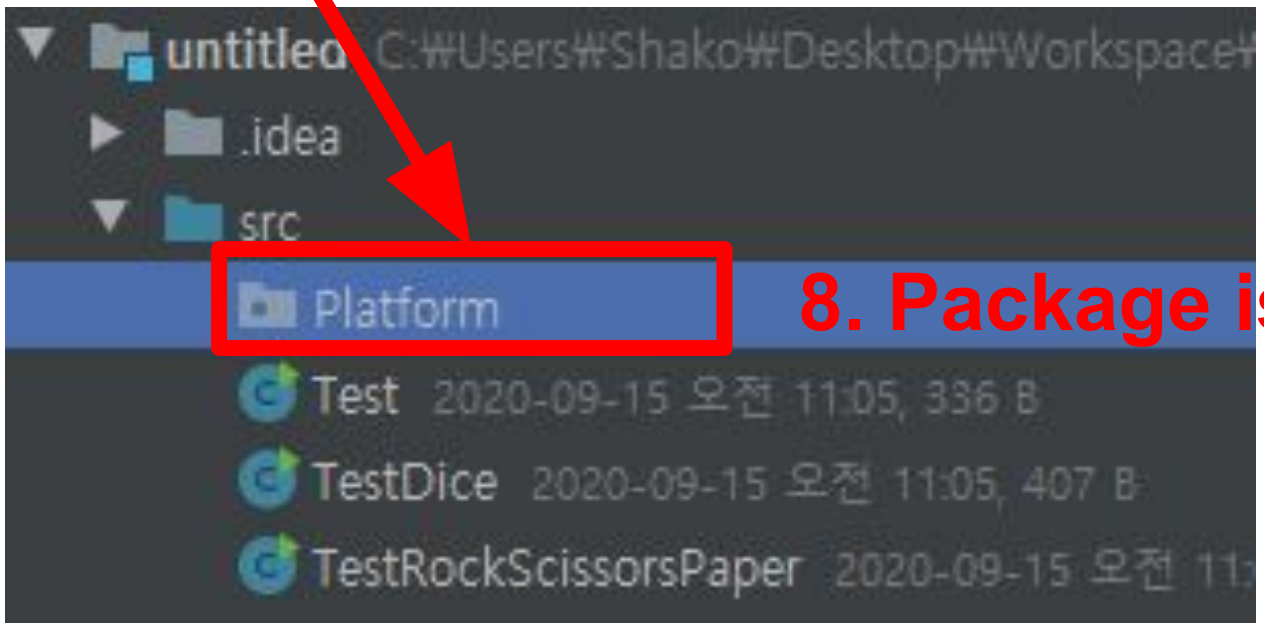
6. Create a new package



Create Packages



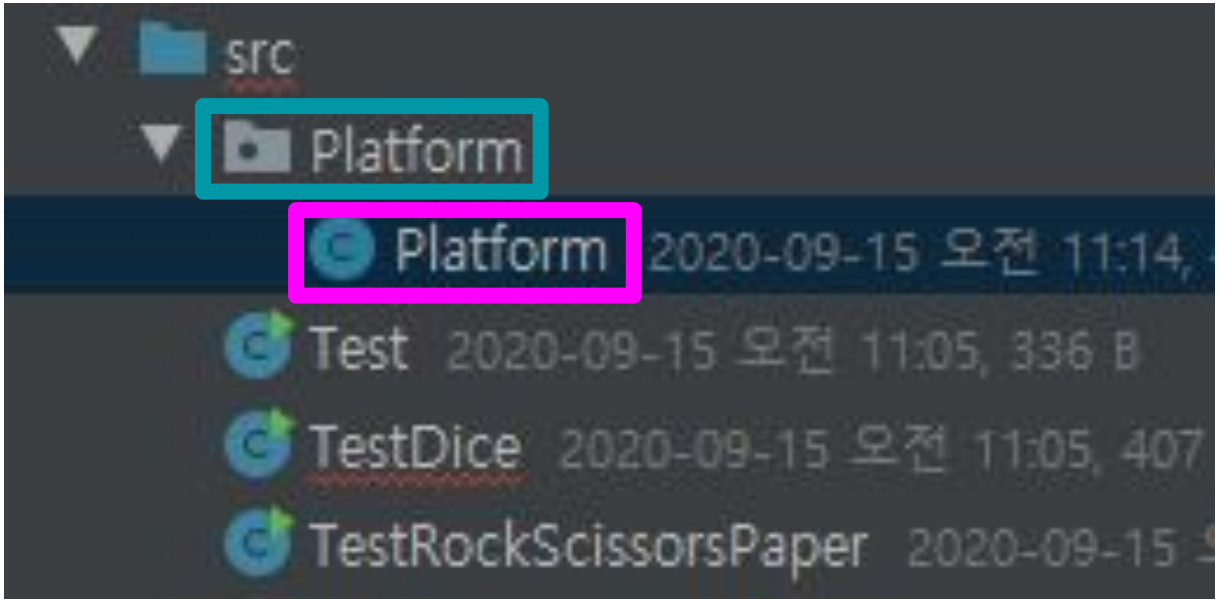
7. Enter the package name



8. Package is created!!

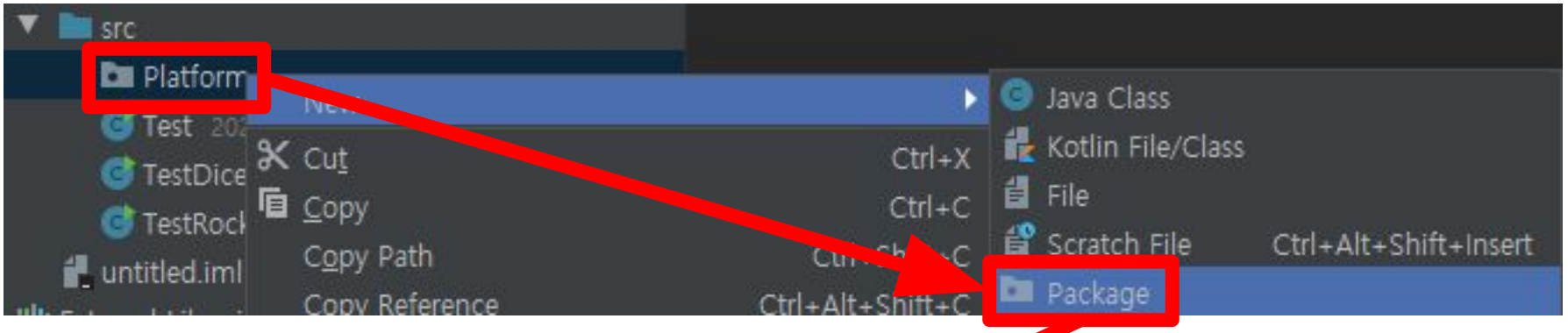
Create Class under the New Package

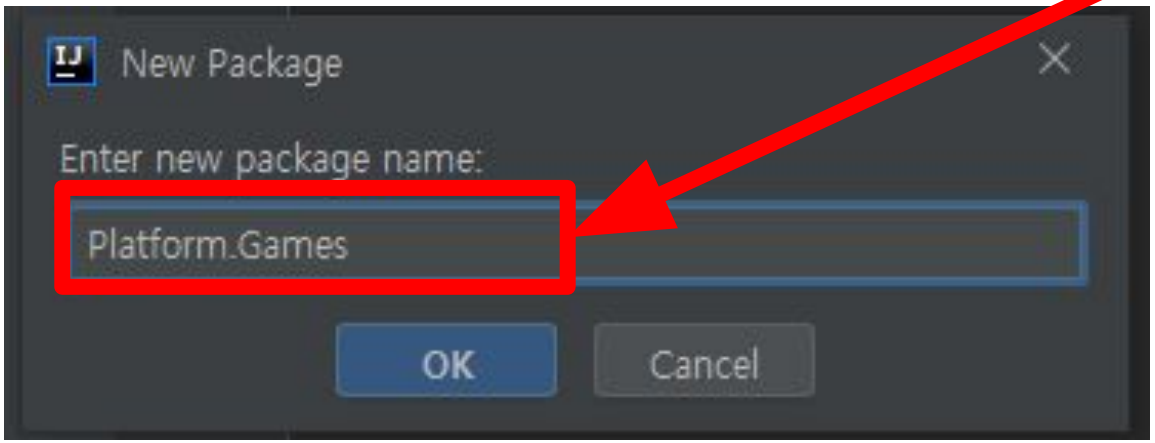
Create "Platform class"
under the "Platform Package"



Create Package under the Package

- We can create more packages under the package!

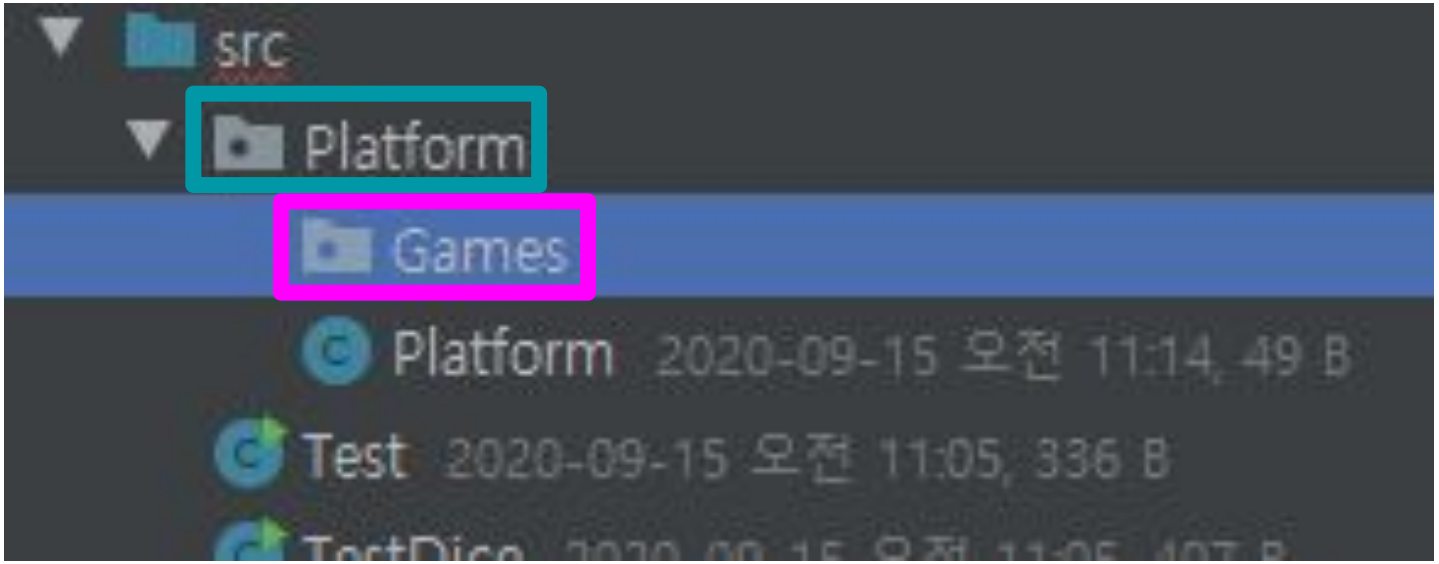




Create Package under the Package

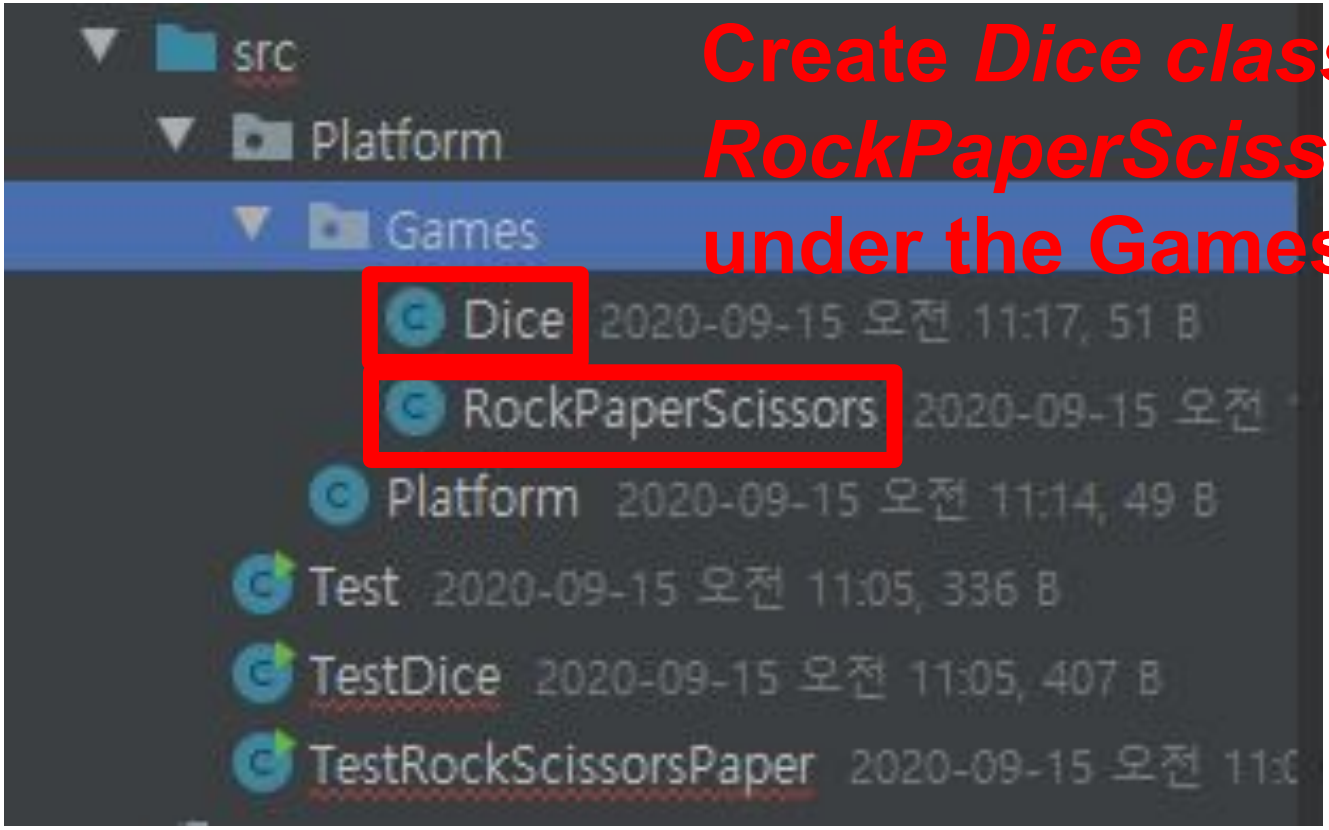
- We can create more packages under the package!

New package (Games Package) is created under the Platform package



Create Class under the Package

- We can also create class under the package under the package!



Create *Dice class* and *RockPaperScissors* class under the Games package

5 mins

Let's Build the Layout!

- Please make the package and classes that you created as below.

```
public class Dice {
    public int playGame() { return -1; }
}
```

```
public class RockPaperScissors {
    public int playGame() { return -1; }
}
```

```
public class Platform {
    public float run(){ return -0.0f; }
    public void setRounds(int num) { }
}
```

Goal of the Problem 1

- Understand the concept of packages.
- Use already implemented functionality (API)
  - Math.random, Scanner

10 mins (5 mins Q&A)

Problem 1. Implement Dice Class

- Objective: Implement `public int playGame` method of `Dice`.
- Description of the method
  - The dice randomly outputs int from 0 to 99.
    - Use `Math.random()` function wisely.
  - The user and the opponent rolls the dice one time.
    - If user's dice int is large than the opponent(win), return 1.
    - If user's dice int is smaller than the opponent(lose), return -1.
    - else (draw), return 0.
  - Before returning, `println` the user's int and the opponent's int with **a single space** in between them.
    - ex1) `47 11` ex2) `40 42`
  - You can test it with the `TestDice` class in the skeleton.

Dice Class Skeleton

```
package Platform.Games;
public class Dice {
    public int playGame() {
        // TODO Lab 4. Problem 1.
    }
}
```

Goal of the Problem 2

- Implement the class with similar interface.
  - Concept of abstraction is needed!

10 mins (5mins Q&A)

Problem 2. RockPaperScissors Class

- Objective: Implement `public int playGame` method of `RockPaperScissors` class.
- Description of the method:
  - User console inputs one of words “scissor”, “rock”, “paper”
    - Beware of the case. Yes, it should be case sensitive. If the user writes a word other than these three words, the user loses. (return -1)
  - opponent randomly choose one of “scissor”, “rock”, “paper”.
    - Use `Math.random()` function wisely.
  - User & Opponent does the Rock Paper Scissors game.
    - If User wins, return 1. If draw, return 0. Else, return -1.
  - Before returning, `println` the user’s choice and the opponent’s choice with **a single space** in between them.
    - ex1) `scissor paper` ex2) `rock paper`
  - Test it with the `TestRockScissorsPaper` class in the skeleton.

RockPaperScissors Class Skeleton

```
package Platform.Games;

public class RockPaperScissors {
    public int playGame() {
        // TODO Lab 4. Problem 2.
    }
}
```

Goal of the Problem 3

- Understand Inter-package import
- Understand the necessity of getter/setter
- Understand the necessity of the generalization of the class. (Will be presented in Inheritance)

Problem 3. Implement Platform Class

- Objective: Implement `public float run` and `public void setRounds` method of `Platform` class.
- Description of the method:
  - `setRounds` sets a number of game rounds per `run()` call.
    - After the first call of this, the consequent setRounds **should not** be able to set the game rounds.
    - The initial number of rounds should be 1.
  - `run()` first console inputs an integer 0 or 1.
    - The number of rounds to play should be set randomly between 5 ~ 10.
    - if 0, play `Dice.playGame` for the number of rounds `setRounds` has set, and return the win rate in **float**.
    - if 1, play `RockPaperScissors.playGame` for the number of rounds `setRounds` has set, and return the win rate in **float**.
    - Win rate : (number of wins) / (total number of rounds)
  - You can test it with the `Test` class in the skeleton.

Problem 3. Implement Platform Class

- Expected Console I/O of the Test class

ex1

```
Choose 0 for Dice, 1 for
RockPaperScissors
0
73 38
58 10
95 26
69 39
2 65
38 77
0.6666667
```

ex2

```
Choose 0 for Dice, 1 for
RockPaperScissors
1
scissor
scissor scissor
paper
paper rock
rock
rock rock
paper
paper scissor
scissor
scissor paper
scissor
scissor rock
0.33333334
```

ex3

```
Choose 0 for Dice, 1 for
RockPaperScissors
1
scissor
scissor paper
paper
paper rock
paper
paper scissor
rock
rock paper
rock
rock paper
0.33333334
```

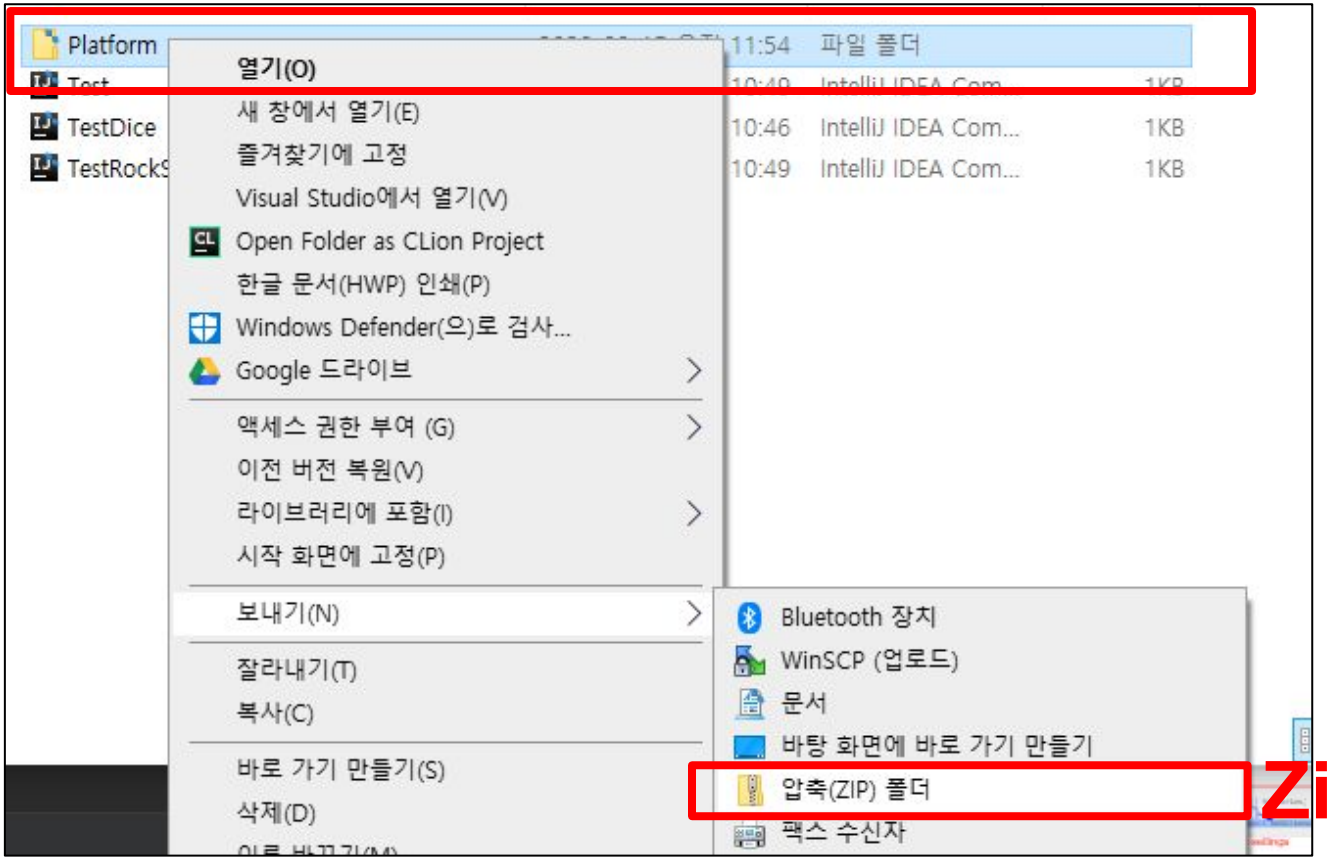
Platform Class Skeleton

```
package Platform;
public class Platform {
  public float run(){
    // TODO Lab 4. Problem 3.
  }
  public void setRounds(int num) {
    // TODO Lab 4. Problem 3.
  }
}
```

Submission

- Compress your `Platform` package directory into a zip file.
  - After unzip, the 'Platform' directory must appear.
- Rename your zip file as 20XX-XXXXX_{name}.zip - for example, 2021-12345_HyunaSeo.zip
- Upload it to eTL - Lab 4 assignment.
- Your program should contain `Platform.Platform` class, `Games.Dice` and `Games.RockPaperScissors` that does not prompt compile error on the skeleton code.

Submission



Zip it

Thank You!!!

Q&A Time!