

# Object – Oriented Programming

---

LAB #4. CLASS 2

# Static

---

- 클래스는 고유의 데이터와 Method를 가질 수 있다.
- new 키워드로 생성된 객체가 아닌 Class에 고정된다.

\* 추가 설명: <https://kingofbackend.tistory.com/131>

ex) 직원 클래스는 생성된 개체 수에 대한 카운트를 유지할 수 있다.

```
public class Employee {  
    public static int empCount;           //not an instance variable  
    public String name;  
    public int age;  
    public int salary;  
  
    public Employee(String name, int age, int salary) {  
        this.name = name;  
        this.age = age;  
        this.salary = salary;  
        empCount++;                       //increment the number of employees  
    }  
}
```

# Static

---

- Employee.empCount 를 호출하여 해당 클래스의 static 변수를 사용한다.
- Static 변수를 호출할 때에는 [클래스 명].[변수 명] 형태로 호출할 수 있다.

```
public class EmployeeManager {  
    public static void main(String[] args) {  
        Employee emp1 = new Employee("James Wright", 42, 20000);  
        Employee emp2 = new Employee("Amy Smith", 27, 8000);  
        Employee emp3 = new Employee("Peter Coolidge", 32, 12000);  
        Employee emp4 = new Employee("John Doe", 22, 10000);  
  
        System.out.println(emp1);           System.out.println(emp2);  
        System.out.println(emp3);           System.out.println(emp4);  
  
        System.out.println("Number of Employees: " + Employee.empCount);  
    }  
}
```

# Static

---

- JAVA 에서는 다음 항목들을 static 으로 선언할 수 있다.

1) static variable

○ ex) `static int empCount;`

2) static method

○ ex) `public static double area(double radius){ ... }`

3) static classes

○ 일반적으로 static 내부 클래스(inner class)와 함께 사용된다.

○ 추후에 다시 자세히 설명

참고링크: <https://siyoon210.tistory.com/141>

# Static Variables

---

- 정적 변수는 클래스 전체에 속하는 변수이며, 하나의 객체에만 속하지 않는다.

각 객체가 다른 데이터를 갖는 Instance 변수와는 달리 정적 변수는 하나의 데이터만 갖는다.

- Class의 모든 객체가 정적 변수를 읽고 변경할 수 있다.

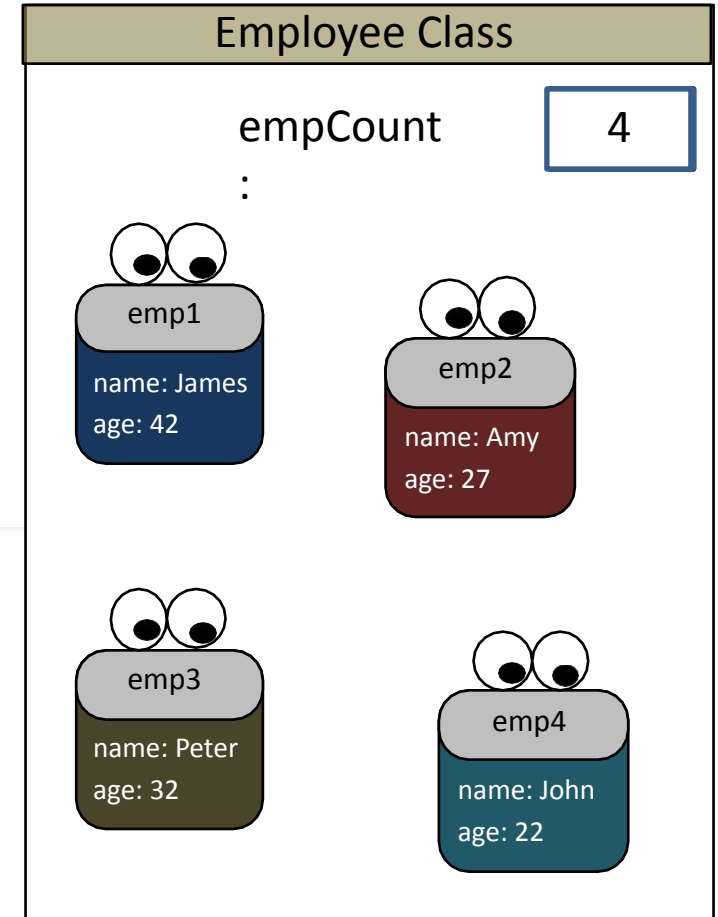
- 정적 변수는 Instance 변수처럼 선언되고 `static` 이라는 키워드가 추가된다.

ex) `private static int myStaticVariable;`

System 클래스의 `out` 도 static variable(field), `new` 키워드 없이 사용 가능.

# Static Variables

```
public class EmployeeManager {  
    public static void main (String[] args){  
        Employee emp1 = new Employee("James Wright",42,"Manager", 20000);  
  
        Employee emp2 = new Employee("Amy Smith",27,"Design Coordinator", 8000,15);  
        Employee emp3 = new Employee("Peter Coolidge",32,"Assistant Manager", 12000,7);  
        Employee emp4 = new Employee("John Doe",22,"Engineer", 10000,10);  
  
        System.out.println(emp1.toString()+emp2+emp3+emp4);  
        System.out.println("Number of Employees: "+Employee.empCount +"\n");  
    }  
}  
  
public Employee(String name, int age,String position, int salary){  
    this.name = name;  
    this.age = age;  
    this.position = position;  
    this.salary = salary;  
    this.vacationDays = 20;  
    empCount++;  
}
```



# Static Method

---

- 정적 Method : 객체를 호출하지 않고 사용하는 Method
- 정적 Method를 정의하기 위해선 Method Signature에 static 키워드를 추가한다.
- 정적 Method는 Instance 변수에 접근할 수 없지만 정적 변수에는 접근할 수 있다.

ex) `public static int myMethod() { ... }`  
`int num = MyClass.myMethod();`

# Static Method

```
/**
 * Class with static methods for circles and spheres.
 */
public class RoundStuff
{
    public static final double PI = 3.14159;

    /**
     * Return the area of a circle of the given radius.
     */
    public static double area(double radius)
    {
        return (PI*radius*radius);
    }

    /**
     * Return the volume of a sphere of the given radius.
     */
    public static double volume(double radius)
    {
        return ((4.0/3.0)*PI*radius*radius*radius);
    }
}
```

```
import java.util.Scanner;

public class RoundStuffDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter radius:");
        double radius = keyboard.nextDouble();

        System.out.println("A circle of radius "
            + radius + " inches");
        System.out.println("has an area of " +
            RoundStuff.area(radius) + " square inches.");
        System.out.println("A sphere of radius "
            + radius + " inches");
        System.out.println("has an volume of " +
            RoundStuff.volume(radius) + " cubic inches.");
    }
}
```



# Main Method

---

- Main Method 는 프로젝트가 실행될 때 가장 먼저 실행되는 Method
- 모든 클래스에 Main Method를 추가할 수 있다.

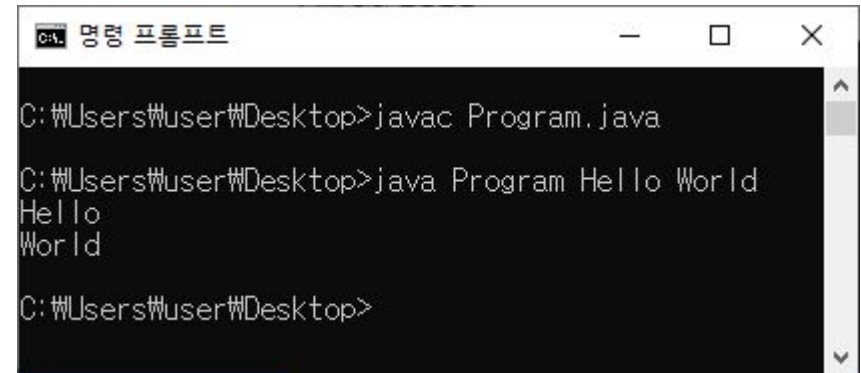
```
public static void main(String[] args) {  
    ...  
}
```

# String args[]

---

- Main Method의 매개변수
- 응용 프로그램을 실행되면서 Runtime System은 Command-Line 인수들을 여러 String 배열을 통해 응용 프로그램의 Main Method에 전달한다.

```
public class Program {  
    public static void main(String[] args) {  
        for(int i = 0; i < args.length; i++) {  
            System.out.println(args[i]);  
        }  
    }  
}
```

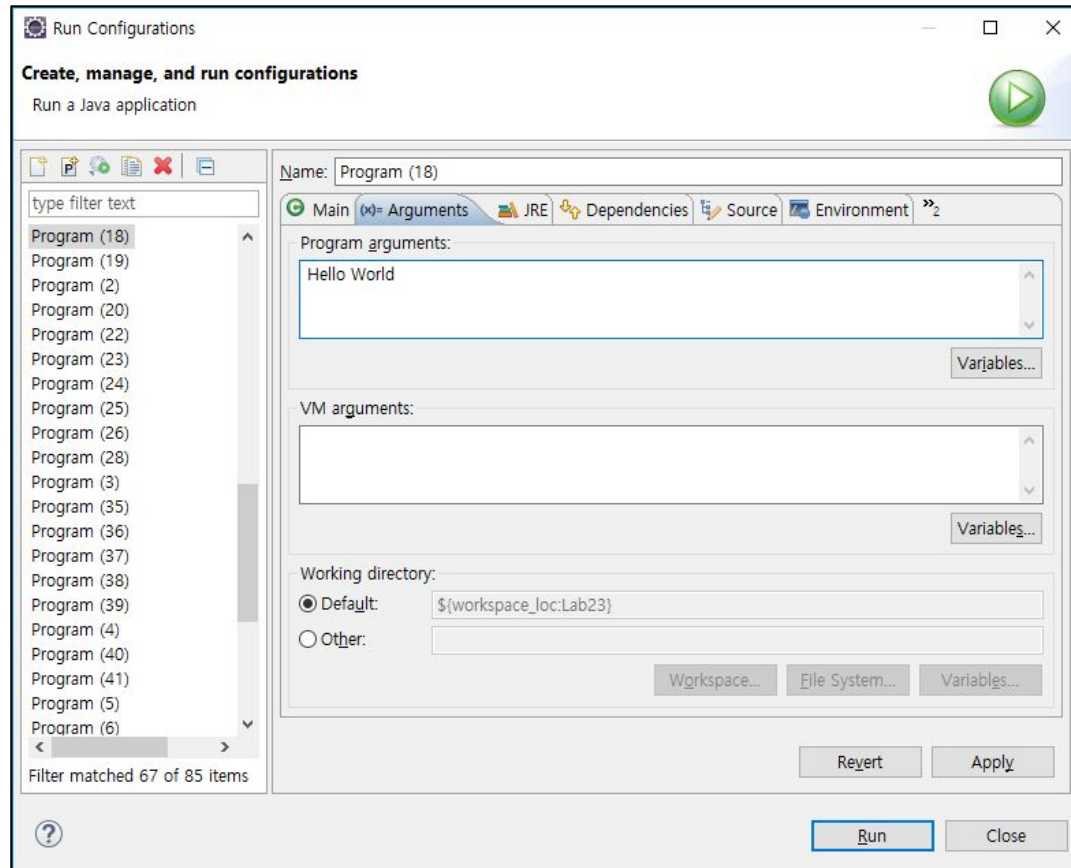


A screenshot of a Windows Command Prompt window titled "명령 프롬프트". The window shows the following commands and output:

```
C:\Users\User\Desktop>javac Program.java  
C:\Users\User\Desktop>java Program Hello World  
Hello  
World  
C:\Users\User\Desktop>
```

# String args[]

- 이클립스 내의 [Run] -> [Run Configurations] 에서 Arguments를 입력할 수 있다.



# The Math Class

---

- Math Class는 static 변수와 static Method들을 갖고 있다.
- 일반적인 산술에 관련된 함수들을 갖고 있다.
- java.lang 패키지에 들어있는 Class이며 별도의 import문이 필요하지 않다.
- ex) PI, E, 등

# The Math Class

---

```
public class Math{  
    public static final double PI =  
        3.141592653589793;  
    public static double sin(double d){ .. }  
    public static double sqrt(double d) { .. }  
  
    private Math(){}  
    ..  
}
```

```
>Math.PI
```

```
3.141592653589793
```

```
>Math.sqrt(25)
```

```
5.0
```

# The Math Class

---

- Math Class는 일반적인 클래스와 달리 객체를 생성할 필요가 없다.
  - Math Class는 Stateless Class이다.
  - 인스턴스화할 필요가 없다.(new를 사용하여 객체 생성할 필요 없음)
  - 하나의 Math Class만 있으면 사용 가능하다.
  - Math Class 내의 모든 변수와 Method는 정적이다.

# The Math Class

---

```
public class MathTest {  
    public static void main(String[] args) {  
        System.out.println("2의 10승 : " + Math.pow(2,10));  
        System.out.println("-5의 절대값 : " + Math.abs(-5));  
        System.out.println("5와 2중 큰 수 : " + Math.max(5, 2));  
        System.out.println("1 ~ 10 까지의 난수 : " + (Math.random() * 10) + 1);  
    }  
}
```

# The Math Class

Math 클래스의 메소드	설명
<code>double pow(double x, double y)</code>	
<code>double sqrt(double x)</code>	
<code>double sin(double x)</code> <code>double cos(double x)</code> <code>double tan(double x)</code>	
<code>int abs(int x)</code> ( <code>long</code> , <code>float</code> , <code>double</code> )	
<code>int min(int x, int y)</code> ( <code>long</code> , <code>float</code> , <code>double</code> )	x와 y값 중 작은 값을 반환
<code>int max(int x, int y)</code> ( <code>long</code> , <code>float</code> , <code>double</code> )	x와 y값 중 큰 값을 반환
<code>double random()</code>	0.0 ~ 1.0 사이의 난수를 반환



# Random Numbers (1)

---

- Math Class에는 난수를 생성하는 Method를 제공한다.
  - `public static double random()`
- random Method의 반환 값은 0.0 ~ 1.0 에 포함되는 모든 실수이다.

ex) `double random = Math.random();`

- 1~100 범위에 있는 난수 생성
  - `double random = (int)(Math.random() * 100) + 1;`

# Random Numbers (2)

---

– Math Class가 아닌 Random Class를 사용하여 난수 생성

1) 해당 클래스를 import

```
import java.util.Random;
```

2) Random 클래스 객체 생성

```
Random rnd = new Random();
```

또는,

```
Random rnd = new Random(seed);
```

# Random Numbers (2)

---

3) 0 ~ 9 사이의 랜덤 정수값 생성

```
int r_num = rnd.nextInt(10);
```

4) 0 ~ 1 사이의 랜덤 실수값 생성

```
double d = rnd.nextDouble();
```

# Wrapper Classes

---

- 기본 데이터 반환형 타입을 포장하는 클래스
- 기본형 타입(int, char, 등)이 객체형으로 사용되어야 하는 경우에 사용

기본형	Wrapper 클래스
byte	Byte
boolean	Boolean
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character

# Boxing & Unboxing

---

- Boxing 은 기본형을 Wrapper class의 객체로 포장하는 과정
- 생성자를 사용하여 매개변수로 변수 값을 전달한다.

```
Integer integerObj = new Integer(42);
```

```
Double doubleObj = new Double(3.2);
```

```
Boolean boolObj = new Boolean(true);
```

- 자동으로 Boxing을 지원한다.

```
Integer integerObj = 42;
```

```
Double doubleObj = 3.2;
```

```
Boolean boolObj = true;
```

# Boxing & Unboxing

---

- Unboxing 은 Wrapper Class의 객체를 기본형으로 해제하는 과정
- [typename]Value() Method를 사용하여 Unboxing 할 수 있다.

```
int i = integerObj.intValue();  
double d = doubleObj.doubleValue();  
boolean b = booleanObj.booleanValue();
```

- 자동으로 Unboxing을 지원한다.

```
int i = integerObj;  
double d = doubleObj;  
boolean b = booleanObj;
```

# Strings & Wrappers

---

- 각 Wrapper 클래스에는 string 형식을 해당 클래스 타입으로 변환해주는 `parse[type()]` Method가 있다.

ex)

```
String str = "42";  
int num = Integer.parseInt(str);
```

# Strings & Wrappers

---

- toString() : parse[Type] 과는 반대로 [Type] -> String 으로 변환하는 Method

```
int i = 1234;  
String str = Integer.toString(i);  
  
double total = 44;  
String total2 = Double.toString(total);  
  
String str = "1234";  
int num = Integer.parseInt(str);
```



# Wrapper Classes

```
import java.lang.Math;

public class class0405{

    public static void main(String[] args)
    {
        int a = 1;
        int b = 1;

        Integer c = new Integer(1);
        Integer d = 1;

        System.out.println(a==b);
        System.out.println(a==c);
        System.out.println(c==d);
        System.out.println(c.equals(d));
    }
}
```

Problems @ Javadoc Declaration Console X

<terminated> class0405 [Java Application] C:\Program Files\Java\W

true  
true  
false  
true

c==d 는 같은 객체인지 확인  
c.equals(d) 는 c와 d의 값이 같은지 확인

<https://coding-factory.tistory.com/536> (설명 참고)

# 실습

---

## City.java

1. City 라는 클래스를 만든다. City 클래스는 3개의 instance 변수를 갖는다.  
(private String name, private int locationX, private int locationY)
2. 모든 instance 변수의 값을 set 하는 생성자를 생성한다.
3. name 값만을 set하는 다른 생성자를 생성한다.  
(이 경우, locationX와 locationY 값은 0~360의 정수로 set 한다. 0포함, 360포함)
4. 각 instance 변수 값을 반환하는 getter를 생성한다.
5. City 객체가 다른 City 객체와 같은지 비교하는 equals() method를 생성한다.  
(3개의 instance변수의 값이 같으면 같은 객체로 인식하게 할것)
6. City 객체의 name, locationX, locationY를 반환하는 toString() Method를 생성한다.  
ex) `System.out.println(city1);` → `Seoul, 14, 52`
7. 두 City 객체간의 거리를 반환하는 Static method, **distance**를 생성한다.
  - 매개변수 : City 객체 2개, return type: double
  - $distance = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$

# 실습

---

## CityTest.java

1. CityTest 라는 이름을 가진 클래스를 생성한다.
2. 해당 클래스에 Main Method를 추가한다.
3. Main Method에서 다음 4개의 객체를 생성한다.  
(“Seoul”, 23, 45), (“Paris”, 123, 41), (“Racoon City”), (“Mega City”)
  - 생성한 모든 객체의 정보를 출력한다. (toString Method 사용)
4. 다음 도시 사이의 거리를 출력한다.  
(Seoul-Paris, Seoul-Racoon City, Paris-Mega City)
  - 거리를 출력하기 전에 도시 정보를 출력한다.  
ex) Seoul-Paris : 100

# 실습

---

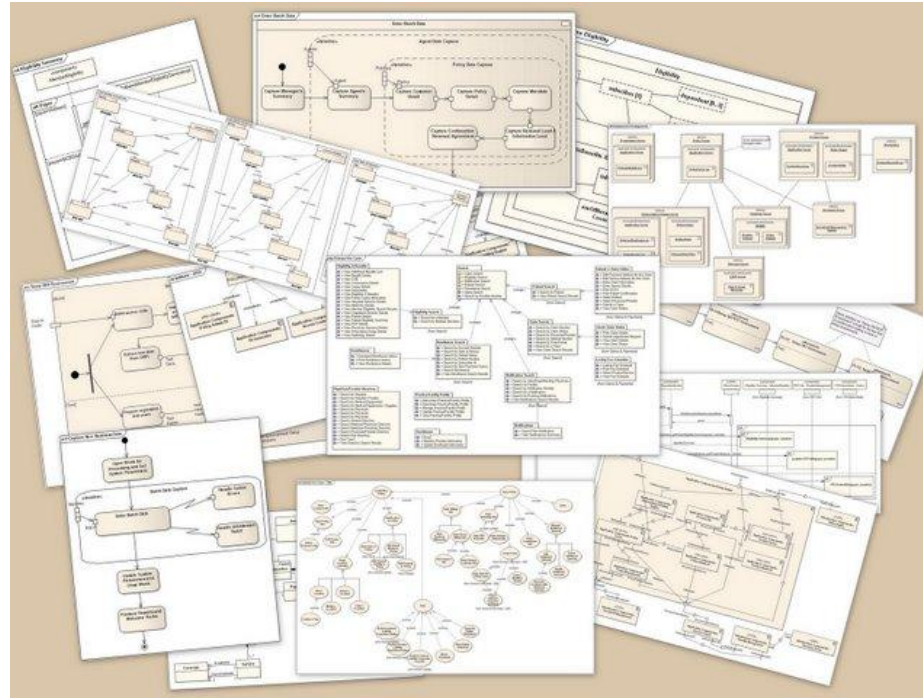
City.java와 CityTest.java를 제출

```
Seoul, 23, 45  
Paris, 123, 41  
Racoon City, 228, 122  
Mega City, 337, 157  
Seoul-Paris : 100.07996802557443  
Seoul-Racoon City : 218.98401768165638  
Paris-Mega City : 243.41733709824368
```

# UML

---

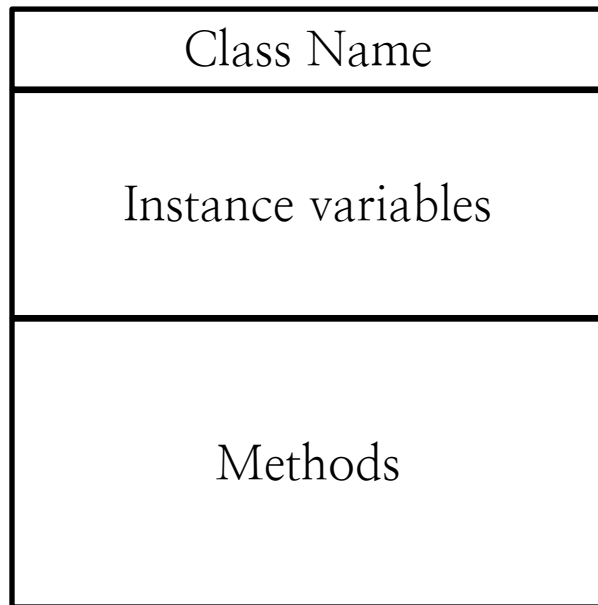
- UML(Unified Modeling Language, 통합 모델링 언어)  
: 객체 지향 프로그래밍 소프트웨어에서 설계, 문서화하기 위해 사용되는 그래픽 언어



# UML

---

## – Class Diagram



### Instance variable:

(modifier) (variable name): (type)

ex) private double side

➔ - side: double

### Method:

(modifier) (method name)((parameters)): (return type)

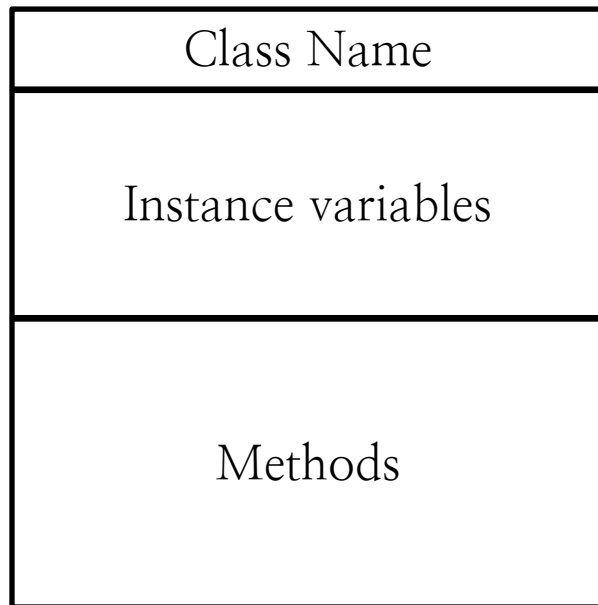
ex) public void reSize(double newSide){...}

➔ + resize(double newSide): void

# UML

---

## – Class Diagram



### Modifier

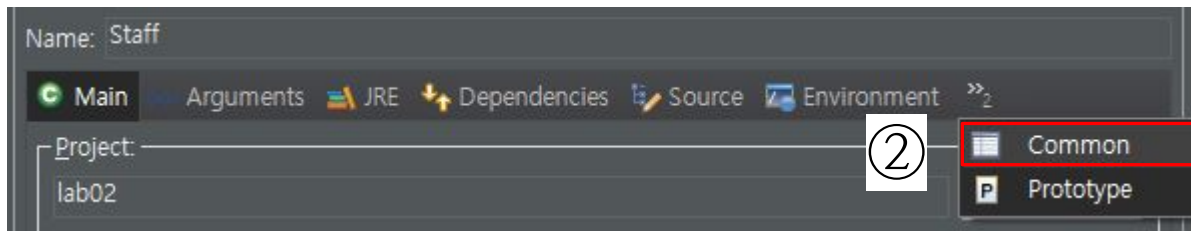
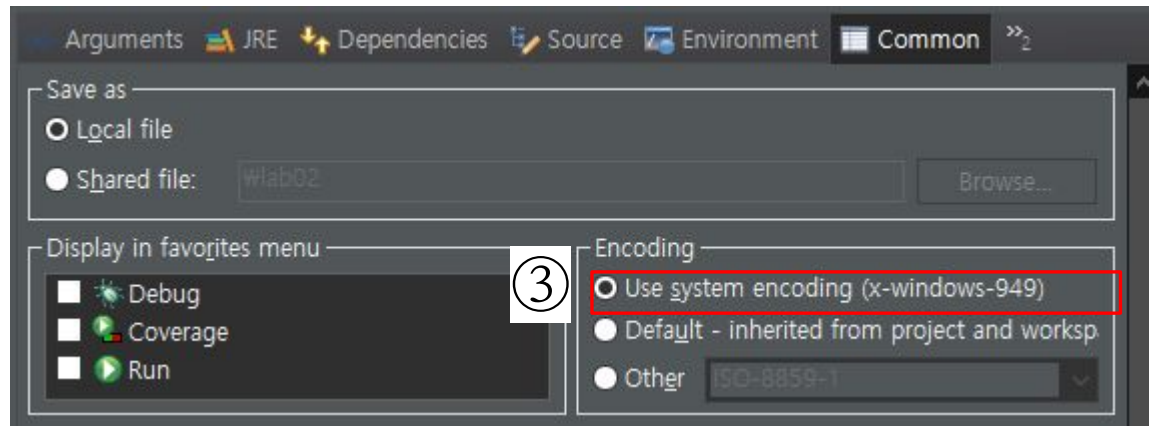
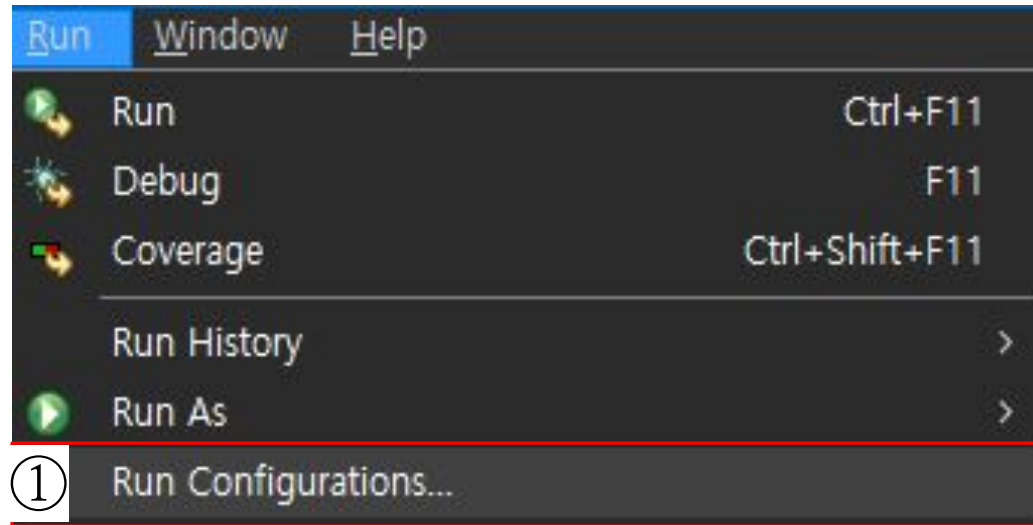
- private: minus(−)
- public: plus(+)
- protected: sharp(#)
- package: tilde(~)
- static: underline
- final: ALL CAPITAL LETTERS

---

Q & A

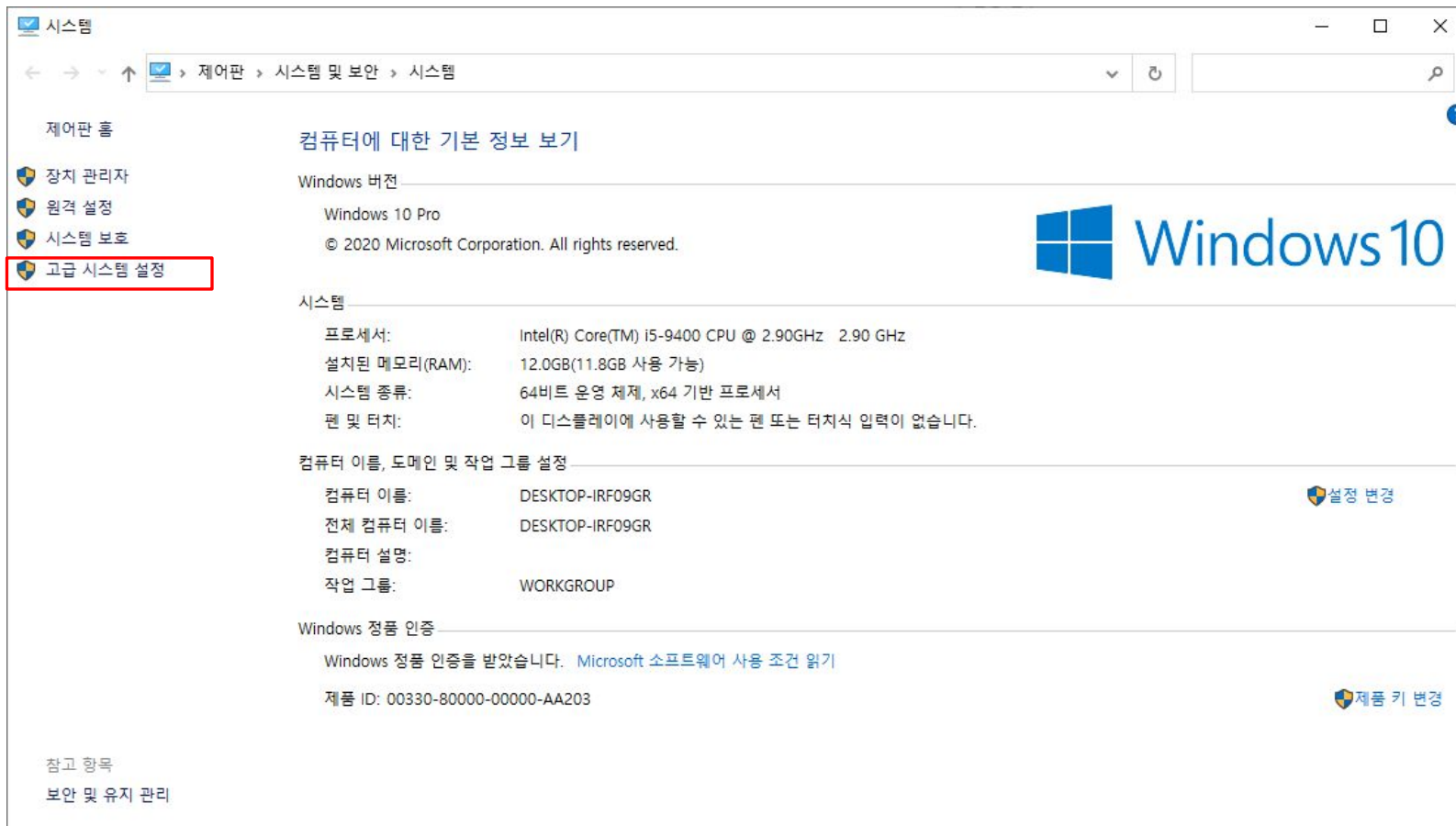


# Console 한글 깨짐



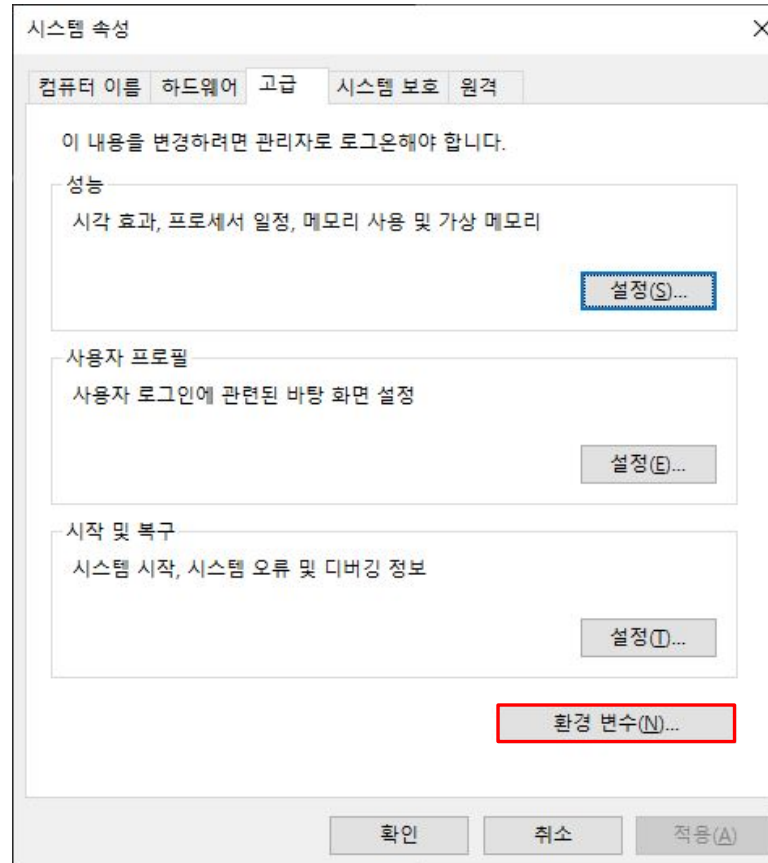
# 환경 변수 설정

## 내PC - 속성 - 고급 시스템 설정



# 환경 변수 설정

## 고급 - 환경 변수



# 환경 변수 설정

---

시스템 변수 - 새로 만들기

- 변수 이름: CLASSPATH

- 변수 값: %JAVA\_HOME%\lib;.

; 세미콜론 뒤에 . (dot ➡ 현재 디렉토리를 의미함) 이 붙어있다는 점을 주의!

