

2

간단한

Java 프로그램

학습 목표

- 자바의 플랫폼 독립성
- 자바 프로그램 기본 구조
- 간단한 자바 프로그램 작성하기

자바의 플랫폼 독립성

PYPL PopularitY of Programming Language

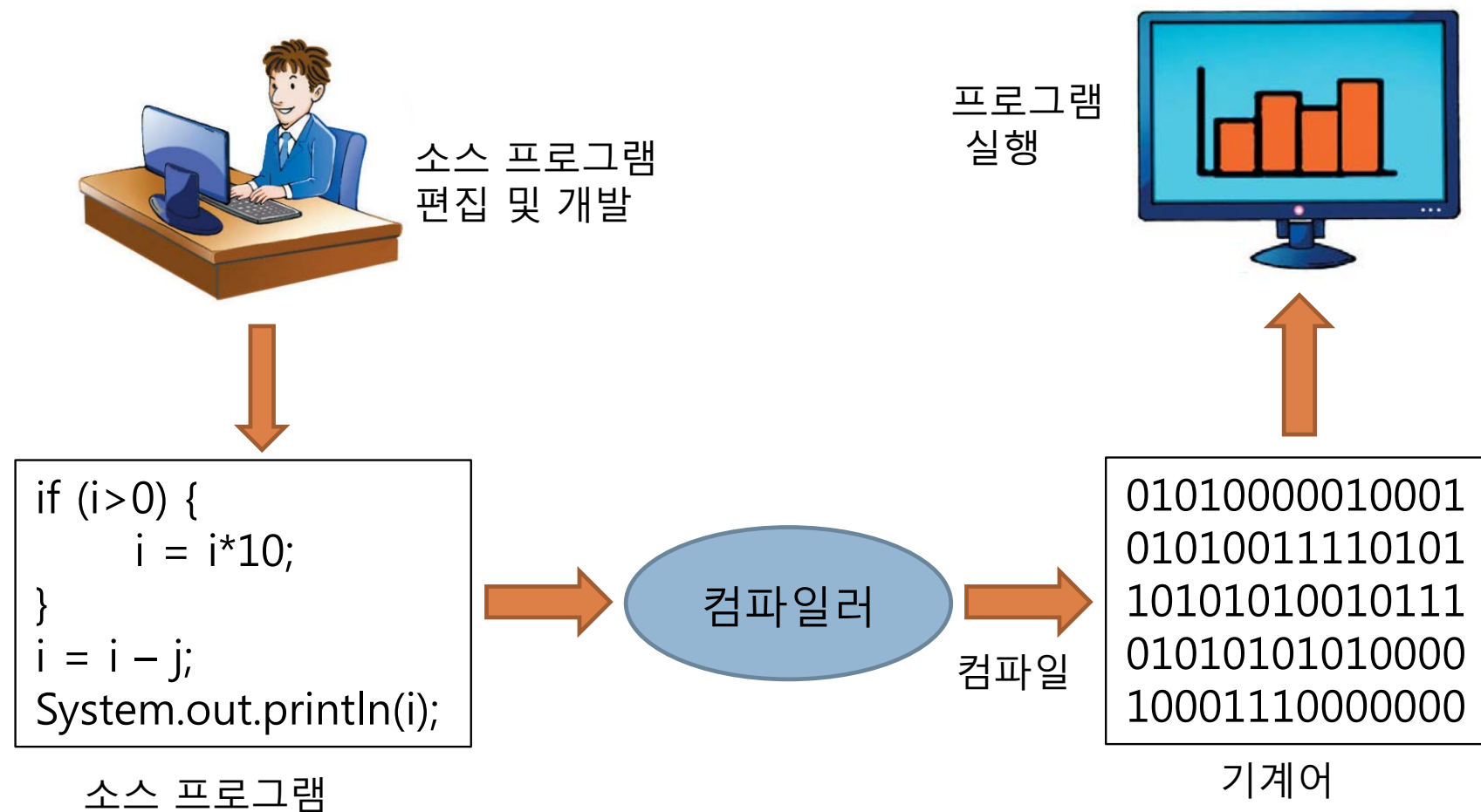
Worldwide, Sept 2018 compared to a year ago:

Rank	Change	Language	Share	Trend
1	↑	Python	24.58 %	+5.7 %
2	↓	Java	22.14 %	-0.6 %
3	↑	Javascript	8.41 %	+0.0 %
4	↓	PHP	7.77 %	-1.4 %
5		C#	7.74 %	-0.4 %
6		C/C++	6.22 %	-0.8 %
7		R	4.04 %	-0.2 %
8		Objective-C	3.33 %	-0.9 %
9		Swift	2.65 %	-0.9 %
10		Matlab	2.1 %	-0.3 %

From <http://pypl.github.io/PYPL.html>

컴파일 (Compilation)

- 소스: 프로그래밍 언어로 작성된 텍스트 파일
- 컴파일: 소스를 기계어로 번역하는 과정



플랫폼 종속성



인텔 CPU를 가진
리눅스 환경에서
개발

C/C++
응용 프로그램

플랫폼 = 하드웨어 플랫폼 + 운영체제 플랫폼

프로그램의 플랫폼 호환성 없는 이유

- 기계어가 CPU마다 다름
- 운영체제마다 API 다름
- 운영체제마다 실행파일 형식 다름

실행

실행되지
않음

실행되지
않음



인텔 CPU + 리눅스



Apple 사의 MAC PC



인텔 CPU + 윈도우 노트북

플랫폼 종속성

- “Hello, world!” 를 출력하는 프로그램 (C program)

```
#include <stdio.h>

void main() {
    printf("Hello, world!\n");
}
```

플랫폼 종속성

○ 어셈블리어 (기계어와 거의 흡사) (Intel Pentium, Linux)

```
.data
msg:
    .ascii "Hello, world!\n"
    len = . - msg

.text
    .global _start
_start:
    movl    $len, %edx
    movl    $msg, %ecx
    movl    $1, %ebx
    movl    $4, %eax
    int     $0x80
    movl    $0, %ebx
    movl    $1, %eax
    int     $0x80
```

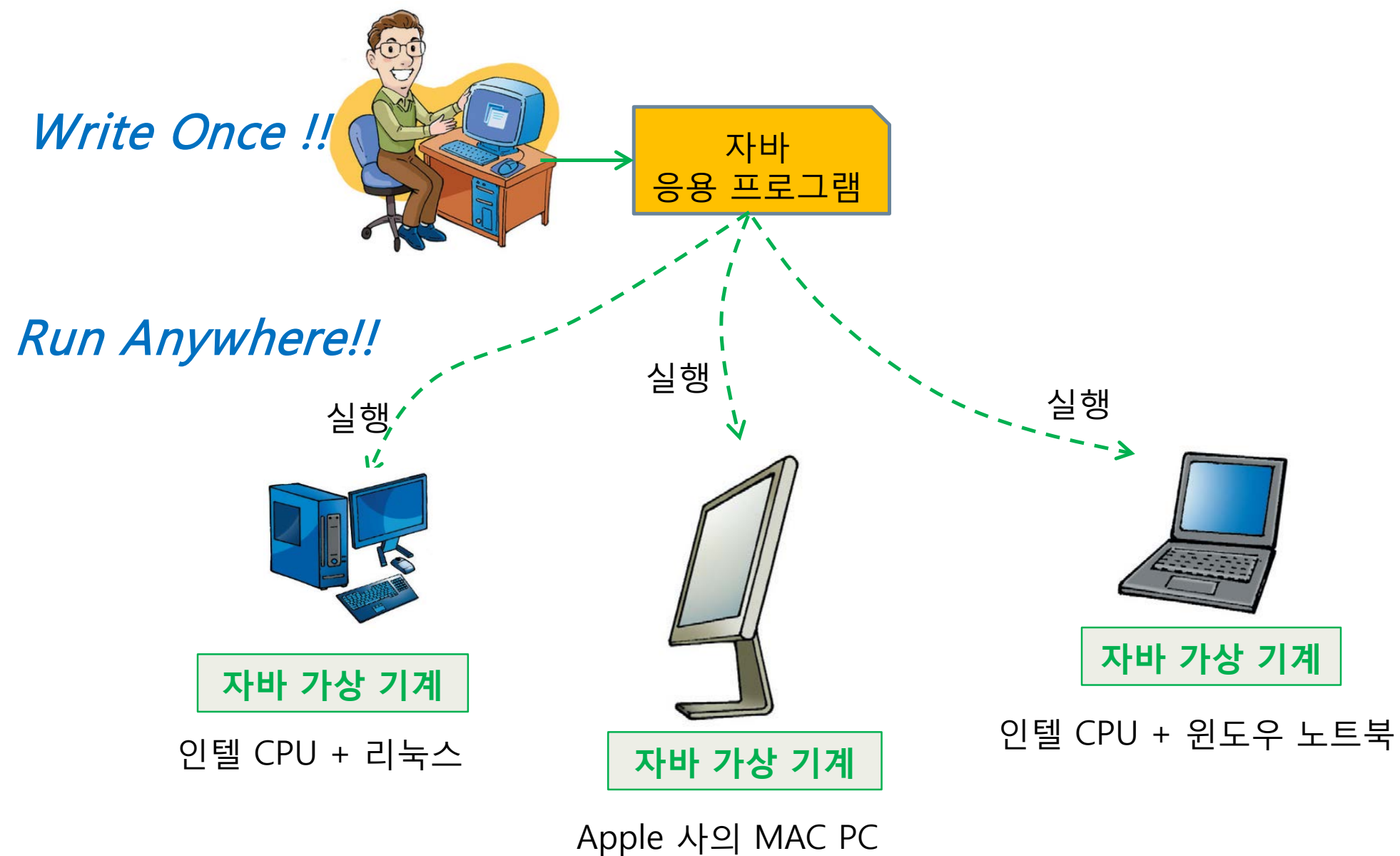

플랫폼 종속성

○ 어셈블리어 (기계어와 거의 흡사) (PowerPC, Linux)

```
.data
msg:
    .string "Hello, world!\n"
    len = . - msg

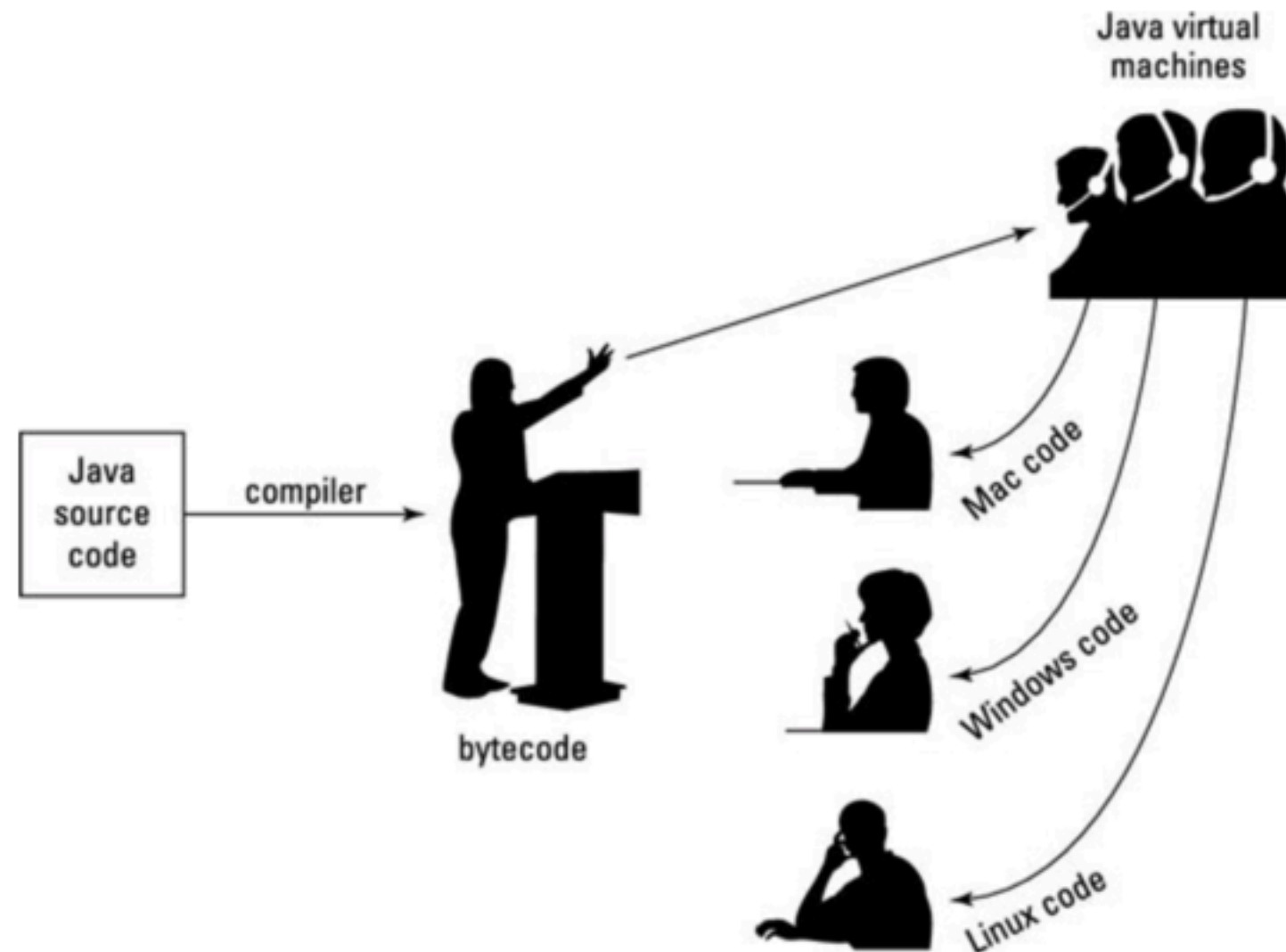
.text
.global _start
_start:
    li      0, 4
    li      3, 1
    lis     4, msg@ha
    addi    4, 4, msg@l
    li      5, len
    sc
    li      0, 1
    li      3, 1
    sc
```

자바의 플랫폼 독립성 (WORA)



자바의 플랫폼 독립성

○ 자바 가상기계 Java virtual machine 이라고 불리는 통역사 덕분!



From <https://www.dummies.com/programming/java/what-is-a-java-virtual-machine/>

자바의 플랫폼 독립성

○ 자바 소스 파일 (Hello.java)

```
public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello to you!")
    }
}
```

자바의 플랫폼 독립성

○ 자바 바이트코드 (Hello.class)

```
.class public HelloWorld
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
    .limit stack 3
    .limit locals 1

    getstatic          java/lang/System/out Ljava/io/PrintStream;
    ldc                "Hello to you!"
    invokevirtual      java/io/PrintStream/println(Ljava/lang/String;)V

    return

.end method
```

질문

- 왜 자바 가상기계는 자바 소스 파일을 직접 실행하지 않고 한단계 변환 과정을 거친 바이트코드를 실행할까?
- 플랫폼 독립성을 보장하는 방식이 플랫폼 종속적인 방식보다 항상 좋을까?

가상의 기계 (Virtual Machine)

- 어떤 언어로 짠 프로그램을 실행하는 프로그램
 - 왜 “기계”? 그 언어가 대개 기계어 수준으로 낮기 때문
 - 왜 “가상”? 그 기계어의 실행기가 하드웨어 (전깃줄)로 손에 잡히지 않고 소프트웨어로 구성되어 있기 때문.

가상의 기계 (Virtual Machine)

- 가상 기계의 용도: 프로그래밍 언어를 구현하는데 번역 compilation 의 징검다리 역할
 - 목표: X 라는 언어를 주어진 컴퓨터의 기계어 Z 로 번역
 - X 를 Z 로 곧바로 번역하기는 그 차이가 너무 큼
 - 중간단계의 언어 Y를 마련: X 에서 Y 로 번역 후 Y 에서 Z 로 번역.
 - 작은 차이를 건너는 번역은 큰 차이를 한 숨에 건너려는 번역 보다는 쉽다.

플랫폼 독립성의 단점

- 컴파일의 이점:
 - 코드 전체를 특정 플랫폼의 기계어로 변환하면 그에 따른 **최적화**가 가능해짐 (여러 명령어 동시에 실행하기 instruction/ data-level parallelism, 메모리 사용 빠르게 하기 cache optimization)
 - 최적화: 어떤 코드를 동일한 작업을 더 효율적으로 수행하는 다른 코드로 바꾸는 일
- 이러한 최적화들은 플랫폼 (예: CPU) 종류마다 다르게 수행되어야 함.

플랫폼 독립성의 단점

- 자바 코드 (.java)를 특정 플랫폼에서 돌 수 있도록 컴파일하면 성능은 빠르나, 플랫폼 종속적
- 자바 가상 기계를 통해서 실행하면 플랫폼 독립적이나 성능은 느림

자바 프로그램의 기본 구조

가장 간단한 예제: Hello

- 다음과 같이 두 줄을 명령어 창에 출력하는 응용 프로그램

```
Hello to you!
```

```
49
```

Hello.java

주석은 `/* */`로 감싼다.

```
/** Hello는 화면에 두 줄을 출력 */
```

```
public class Hello
```

Hello 클래스 정의

```
{
```

```
    public static void main(String[] args)
```

main 메소드 정의

```
{
```

```
    System.out.println("Hello to you!");
```

```
    System.out.println(49);
```

System.out의 println 메소드 호출

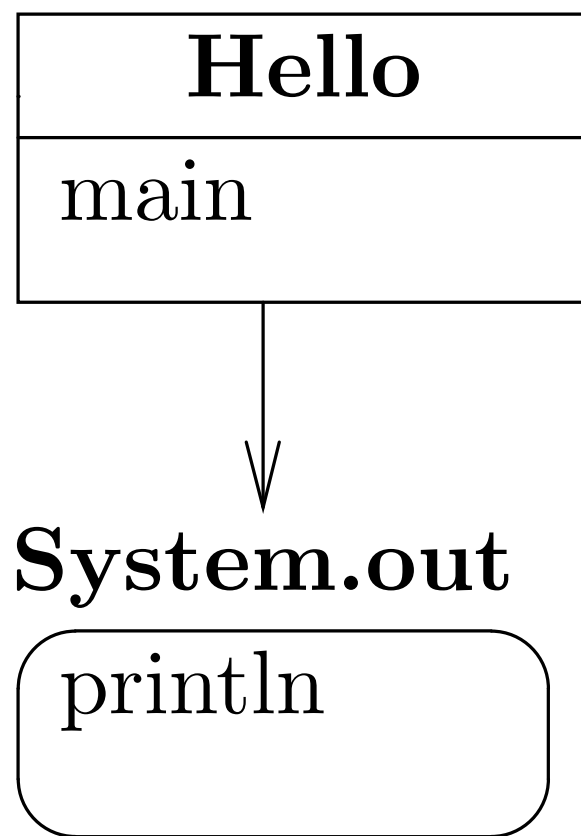
```
}
```

```
}
```

main 메소드

- 프로그램 실행을 시작시키는 메소드 (method)
- 프로그램 객체가 메모리에 생성되고 나서 main 메소드가 호출된다.
- 그러면, main 메소드에 있는 명령어가 실행된다.
- 모든 Java 응용 프로그램(application)은 main 메소드가 있어야 한다.

클래스 구조도



구현해야 하는 것!

명령어 창 객체

어떻게 구현하고 실행하죠?

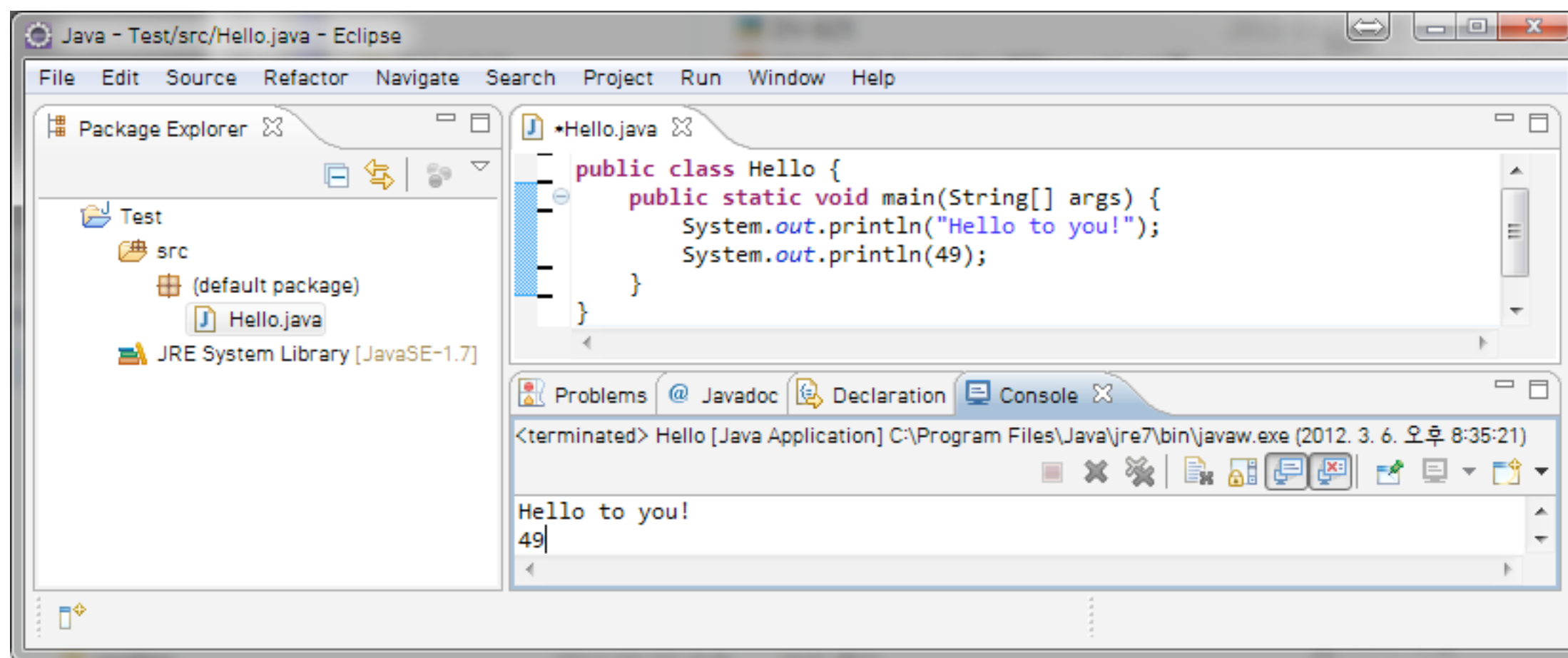
- Hello 클래스를 파일에 입력하고 저장
- 오타는 안된다. 물론, 오타가 있는 경우 컴파일러가 검사해 줄 것이다.
- 컴파일한다.
- 실행시킨다.

자바 통합 개발환경 - 이클립스

- IDE (Integrated Development Environment)란?
 - 통합 개발 환경
 - 편집, 컴파일, 디버깅을 한번에 할 수 있는 개발 환경
- 이클립스 (Eclipse)
 - 자바 응용 프로그램 개발을 위한 통합 개발 환경
 - IBM 에 의해 개발된 오픈 소스 프로젝트

Eclipse 에서는

- 프로젝트 생성 >> 클래스 생성 >> Run
- 알아서 저장하고, 컴파일하고, 실행시켜 준다.



JDK

- JDK (Java Development Kit)
 - 자바 응용 개발 환경, 개발에 필요한 도구 포함 (컴파일러, 클래스 라이브러리, 샘플 등 포함)
- JDK 의 주요 개발 도구
 - javac - 자바 소스를 바이트 코드로 변환하는 컴파일러
 - java - 자바 응용프로그램 실행기
 - jar - 자바 아카이브 파일 (jar)의 생성 및 관리
 - jdb - 자바 디버거

명령어 창으로 직접 할 때는

- Hello.java 작성
- 컴파일: `javac Hello.java` 파일 이름
- 실행: `java Hello` 클래스 이름



```
Pamela:~/class/cse216/2012/java/ — bash — 79x16
Pamela:java oukse$ cat > Hello.java

public class Hello
{
public static void main(String[] args)
{
System.out.println("Hello to you!");
System.out.println(49);
}
}

Pamela:java oukse$ javac Hello.java
Pamela:java oukse$ java Hello
Hello to you!
49
Pamela:java oukse$
```

실행을 따라가 보면

- JVM(Java 가상머신)이 실행된다.
 - System.out과 같은 기본 객체들이 메모리에 있다.
- Hello.java를 컴파일한 파일 Hello.class가 메모리에 복사되면서 Hello 객체가 된다.
 - 사용자가 Hello를 실행하라고 했기 때문에.
- JVM이 main 메소드를 호출한다.

실행을 따라가 보면

① main 메소드 호출

Hello



```
public static void main(String[] args)
{
    System.out.println("Hello to you!");
    System.out.println(49);
}
```

System.out

```
println(x)
{
    x를 화면에
    출력하는 명령들
}
```

실행을 따라가 보면

① main 메소드 호출

Hello



```
public static void main(String[] args)
{
  ② System.out.println("Hello to you!");
  System.out.println(49);
}
```

System.out

```
println(x)
{
  x를 화면에
  출력하는 명령들
}
```

실행을 따라가 보면

① main 메소드 호출

Hello



```
public static void main(String[] args)
{
  ② System.out.println("Hello to you!");
  System.out.println(49);
}
```

③

println
메소드
호출



System.out

```
println(x)
{
  x를 화면에
  출력하는 명령들
}
```


실행을 따라가 보면

① main 메소드 호출

Hello



```
public static void main(String[] args)
{
  ② System.out.println("Hello to you!");
  ④ 메소드 호출이 끝나길 기다림
  System.out.println(49);
}
```

③

println
메소드
호출

System.out

④

실행

```
println(x)
{
  x를 화면에
  출력하는 명령들
}
```



실행을 따라가 보면

① main 메소드 호출

Hello



```
public static void main(String[] args)
{
    ② System.out.println("Hello to you!");
    ④ 메소드 호출이 끝나길 기다림
    ⑤ System.out.println(49);
}
```

③

println
메소드
호출



println
메소드
호출

⑥

System.out

④

```
println(x)
{
    x를 화면에
    출력하는 명령들
}
```

실행



실행을 따라가 보면

① main 메소드 호출

Hello



```
public static void main(String[] args)
{
    ② System.out.println("Hello to you!");
    ④ 메소드 호출이 끝나길 기다림
    ⑤ System.out.println(49);
    ⑦ 메소드 호출이 끝나길 기다림
}
```

③
println
메소드
호출



println
메소드
호출
⑥

System.out

```
println(x)
{
    x를 화면에
    출력하는 명령들
}
```

⑦

실행



종료: main 메소드가 끝났기 때문에

① main 메소드 호출

Hello

```
public static void main(String[] args)
{
    ② System.out.println("Hello to you!");
    ④ 메소드 호출이 끝나길 기다림
    ⑤ System.out.println(49);
    ⑦ 메소드 호출이 끝나길 기다림
}
```

③
println
메소드
호출

println
메소드
호출
⑥

System.out

```
println(x)
{
    x를 화면에
    출력하는 명령들
}
```

⑦
실행

간단한 자바 프로그램 작성하기

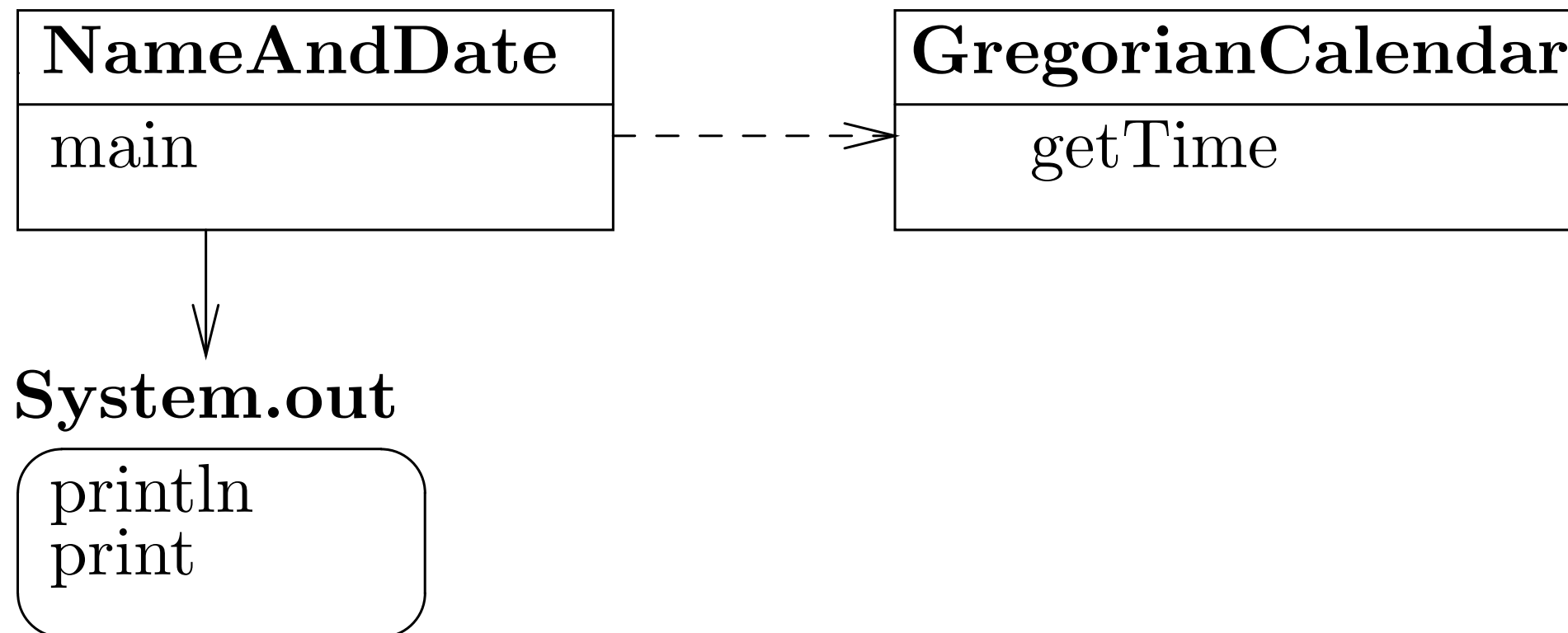
객체가 객체를 생성하는 예제

- 앞 예제는 기존에 존재하는 객체에 메시지 전달
- 이번 예제는 객체를 생성하여 메시지 전달
- 이름, 정확한 날짜, 시간을 출력하는 NameAndDate 프로그램을 작성하자.

```
Hanyang ERICA --- Wed Sep 12 21:32:49 KST 2018  
Finished.
```

- 날짜와 시간을 관장하는 클래스 GregorianCalendar
 - getTime 메소드: 현재 시간을 알려 준다.

클래스 구조도



자주 나오는 소프트웨어 구조: MVC 구조

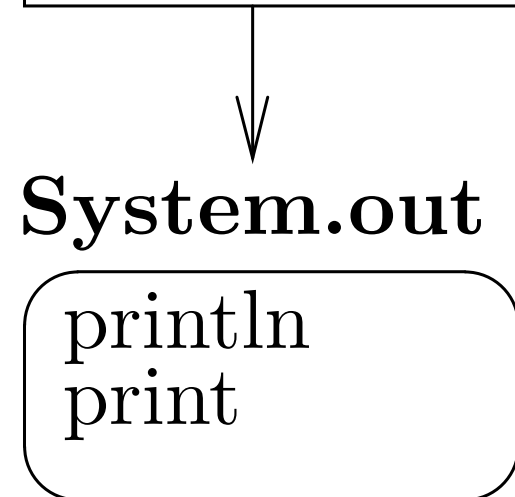
- 모델(model)-뷰(view)-제어기(controller) 구조
 - 뷰 객체: 사용자와의 상호작용을 관장하는 객체
 - 제어기 객체: 정보를 전달하는 것을 관장하는 객체
 - 모델 객체: 제어기의 메시지에 따라 계산을 하는 객체

MVC 구조

제어기 (controller)



모델 (model)



뷰 (view)

NameAndDate.java

GregorianCalendar를 쓰기 위해서

//로 시작하면 한 줄 주석

```
import java.util.*;                                // Java package

/** NameAndDate는 이름과 날짜, 시간을 출력 */
public class NameAndDate
{
    public static void main(String[] args)
    {
        System.out.print("Hanyang ERICA --- ");
        // The next statement creates an object:
        GregorianCalendar c = new GregorianCalendar();
        // c에게 시간을 물어보고 그 결과를 출력      객체 생성은 new로
        System.out.println(c.getTime());
        System.out.println("Finished");
    }
}
```

중요한 구문

- `import java.util.*;`
 - 미리 작성된 패키지를 사용하려면 선언해 주어야 한다.
 - `GregorianCalendar`는 `java.util` 패키지에 있다.
- `new GregorianCalendar()`
 - `new`는 `GregorianCalendar` 클래스로부터 새로운 객체를 생성한다.
 - `()` 안에는 인수(parameter), 즉 객체를 생성하는데 필요한 정보를 줄 수 있으나, 이번 경우에는 인수가 없다.
- `GregorianCalendar c = new GregorianCalendar();`
 - 만들어진 객체에 `c`라는 이름을 준다.
 - `c`가 가지고 있는 객체의 클래스는 `GregorianCalendar`이다.
- `c.getTime()`
 - `c` 객체의 `getTime` 메소드를 호출한다.
- `System.out.println(c.getTime());`
 - 메소드 호출의 결과를 그대로 `System.out`의 `println` 메소드로 전달한다.

실행을 따라가 보면 (중요 부위만)

NameAndDate

```
main
{ System.out.print("Hanyang ERICA --- ");
  GregorianCalendar c = new GregorianCalendar();
  System.out.println(c.getTime());
  System.out.println("Finished");
}
```

System.out

```
print(...)
{ 텍스트 출력 }
println(...)
{ 텍스트와 넘김문자 출력 }
```

실행을 따라가 보면 (중요 부위만)

NameAndDate

```
main
{ System.out.print("Hanyang ERICA --- ");
  GregorianCalendar c = new GregorianCalendar();
  System.out.println(c.getTime());
  System.out.println("Finished");
}
```

① 객체 생성

System.out

```
print(...)
{ 텍스트 출력 }
println(...)
{ 텍스트와 넘김문자 출력 }
```

?? : GregorianCalendar

```
getTime()
{ 시스템 시계를 읽어서 그 값을 반환함 }
```

실행을 따라가 보면 (중요 부위만)

NameAndDate

```
main
{ System.out.print("Hanyang ERICA --- ");
  GregorianCalendar c = new GregorianCalendar();
  System.out.println(c.getTime());
  System.out.println("Finished");
}
```

2 이름 부여

System.out

```
print(...)
{ 텍스트 출력 }
println(...)
{ 텍스트와 넘김문자 출력 }
```

c: GregorianCalendar

```
getTime()
{ 시스템 시계를 읽어서 그 값을 반환함 }
```

실행을 따라가 보면 (중요 부위만)

NameAndDate

```
main
{ System.out.print("Hanyang ERICA --- ");
  GregorianCalendar c = new GregorianCalendar();
  System.out.println(c.getTime());
  System.out.println("Finished");
}
```

③ getTime 호출

c: GregorianCalendar

```
getTime()
{ 시스템 시계를 읽어서 그 값을 반환함 }
```

System.out

```
print(...)
{ 텍스트 출력 }
println(...)
{ 텍스트와 넘김문자 출력 }
```

실행을 따라가 보면 (중요 부위만)

NameAndDate

```
main
{ System.out.print("Hanyang ERICA --- ");
  GregorianCalendar c = new GregorianCalendar();
  System.out.println(c.getTime());
  System.out.println("Finished");
}
```

System.out

```
print(...)
{ 텍스트 출력 }
println(...)
{ 텍스트와 넘김문자 출력 }
```



④
println
호출

c: GregorianCalendar

```
getTime()
{ 시스템 시계를 읽어서 그 값을 반환함 }
```


구문(Syntax)과 의미구조(Semantics)

○ 구문

- 어떻게 생긴 것이 제대로 생긴 것이냐?
- 예, 철자법
- 예, new <클래스이름> (<인수>, ..., <인수>)
- 구문 오류 (syntax error): 구문에 맞지 않게 작성했을 때 컴파일러가 찾아준다.

○ 의미구조

- 구문이 맞는 프로그램이 어떻게 실행될 것이냐?
- 예, 영어 해석법
- 예, new C()는 C 클래스의 객체를 메모리에 생성한다.
- 의미 오류 (semantic error): 실행 중에 그냥 발생한다.
 - 일부는 실행 시간에 검출된다. 예, 타입 오류.

구문을 기술하는 방법

- 일반적으로 BNF(Backus-Naur Form)를 사용

CLASS ::= public class IDENTIFIER { METHOD* }

METHOD ::= public static void METHOD_HEADER METHOD_BODY

METHOD_HEADER ::= IDENTIFIER (FORMALPARAM LIST?)

METHOD_BODY ::= { STATEMENT* }

STATEMENT ::= INVOCATION ;

INVOCATION ::= RECEIVER . IDENTIFIER (ARGUMENT_LIST?)

RECEIVER ::= IDENTIFIER | RECEIVER.IDENTIFIER

- 자세한 내용은 **ELE329** 오토마타와형식언어론, **ENE414** 프로그래밍언어론, **CSE309** 컴파일러구조에서

의미구조를 기술하는 방법

○ 말로 하는 방법

- 예, RECEIVER.IDENTIFIER() 는 RECEIVER가 의미하는 객체의 IDENTIFIER 메소드를 호출한다. 호출된 메소드가 반환하는 결과가 이 계산식의 결과로 한다.
- 애매할 수 있다.

○ 수학으로 하는 방법

- 프로그램의 실행 의미를 수학적 모델을 통해 기술
- 정확하지만 일반 사용자가 이해하기는 쉽지 않다.
- **ENE414 프로그래밍언어론**에 일부 다룬다.

Java 패키지(Package)와 API

- 클래스를 분류하여 패키지화 할 수 있다.
 - 예, erica.util 내에 클래스 MyMap
- Java에서 기본적으로 제공하는 패키지를 Java API (Application Programming Interface)라고 부른다.
 - 예, java.lang, java.util 등
 - 일부 기초적인 API는 import 하지 않아도 사용할 수 있다. 예, System.out.
 - <https://docs.oracle.com/javase/9/docs/api/overview-summary.html> 에 가면 문서화되어 있다.

요약: Java의 기본 구문

- 클래스 정의: `public class Hello { ... }`
- main 메소드: `public static void main (String[] args) { ... }`
- 메소드 호출: `System.out.println(...), c.getTime()`
- 객체 생성: `new GregorianCalendar()`
- 이름 (변수) 생성: `GregorianCalendar c = ...;`