JAVA 입문 : 이론과 실습

JAVA 입문 : 이론과 실습

제 1장 자바의 개요

목차

자바 언어의 소개

- 자바는 범용 프로그래밍 언어이다.
  - 특히, 인터넷 환경과 모바일 환경에 지원
- 자바는 객체지향 언어이다.
  - C++와 유사한 객체지향 언어이면서 복잡한 기능은 제거
- 자바는 플랫폼에 독립적이다.
  - 가상기계코드인 바이트코드를 사용
  - 자바 가상기계(JVM)로 실행

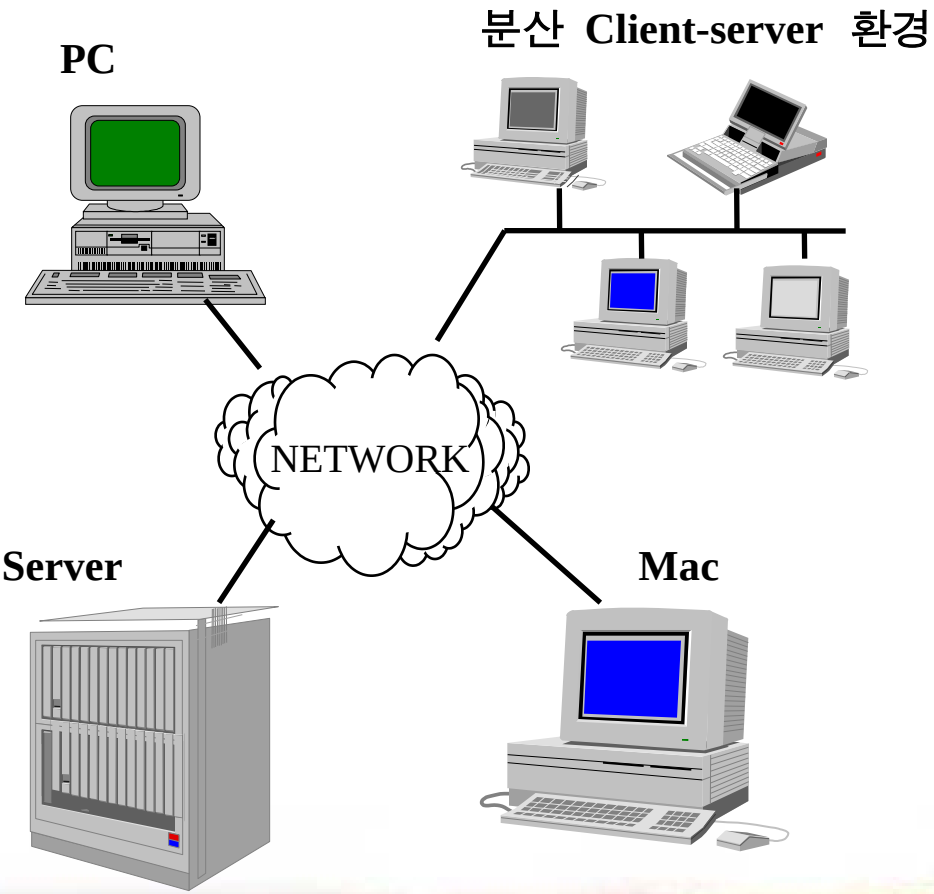
자바 언어의 역사 [1/3]

- Sun Microsystems의 James Gosling이 개발
- 1990년 “그린 프로젝트”로 시작
- 1991년 가전제품을 위한 언어로 개발
- 범용 프로그래밍 언어로 발전

- Java의 어원
  - James Gosling, Arthur Van Hoff, Andy Bechtolsheim
  - 인도네시아 산 커피 원료 이름

자바 언어의 역사 [2/3]

- 인터넷 환경과 모바일 환경에 적합



플랫폼 종속성(platform dependence)

인텔 CPU를 가진
리눅스 환경에서 개발

C/C++
응용 프로그램

▣플랫폼 = 하드웨어 플랫폼 + 운영체제 플랫폼

프로그램의 플랫폼 호환성 없는 이유
- 기계어가 CPU마다 상이함
- 운영체제마다 API기 서로 다름
- 운영체제마다 실행파일 형식이 서로 다름

실행

실행되지 않음

실행되지 않음

인텔 CPU + 리눅스

Apple 사의 MAC PC

인텔 CPU + 윈도우 노트북

자바의 플랫폼 독립성, WORA

Write Once !!

자바
응용 프로그램

Run Anywhere !!

실행

실행

실행

자바 가상 기계

인텔 CPU + 리눅스

자바 가상 기계

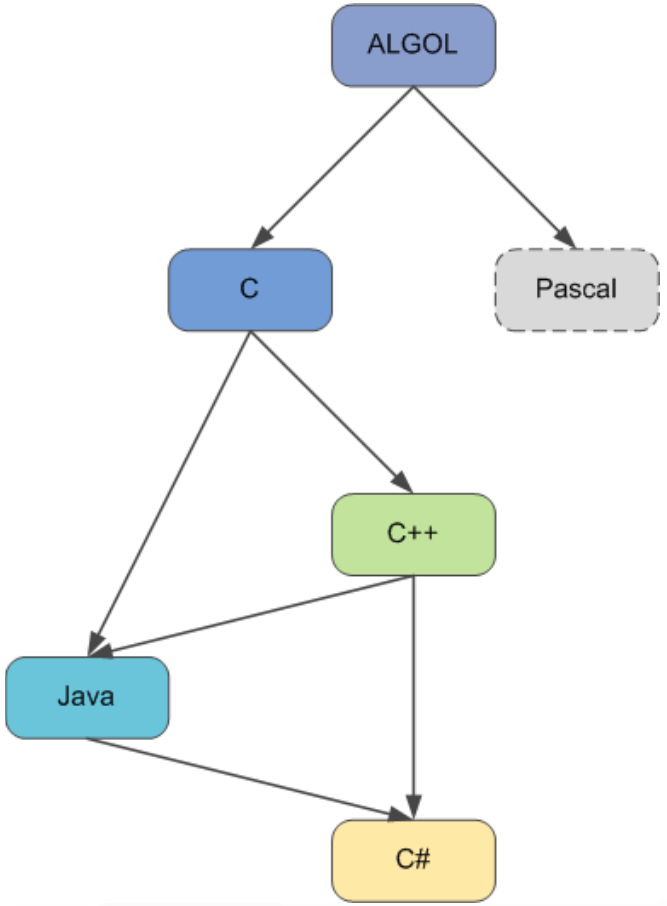
Apple 사의 MAC PC

자바 가상 기계

인텔 CPU + 윈도우 노트북

자바 언어의 역사 [3/3]

- 자바의 계층 구조
  - C 계열의 언어
    - C의 연산자와 문장 등 기초 프로그래밍 언어 기능
    - C++의 객체지향 속성과 예외처리, 제네릭 기능
  - 자바에서 확립된 중요 개념
    - 예외 처리
    - 스레드
    - C#에 영향을 미침



자바의 5가지 주요 특성

1. 간단하며 친숙한 언어
2. 객체지향 언어
3. 플랫폼에 독립적
4. 견고하고 보안에 강하다
5. 동적 링킹과 멀티스레드를 지원

특성 1. 간단하며 친숙한 언어

- 설계 목표 :
  - 작고 간단해서 쉽게 프로그래밍하고, 쉽게 디버깅하고, 쉽게 배울 수 있도록 하는 것
- C/C++의 복잡한 기능 일부 삭제
  - 포인터 연산 삭제, 자동적인 메모리 관리
- 친숙한 언어
  - 기본기능 : 연산자와 문장 구조 - ANSI C(표준 C) 언어
  - 고급기능 : 객체 지향적인 개념 - C++ 언어

C/C++에서 제거된 특성 [1/2]

- typedef문, #define문
  - 클래스와 인터페이스
- 구조체(struct)와 공용체(union)
  - 클래스로 대체 가능
- 함수(function)
  - 모두 메소드로 처리
- 다중상속(multiple inheritance)
  - 인터페이스는 다중상속 지원
- goto 문을 지원하지 않음.
  - 다중 레이블 break/continue문

C/C++에서 제거된 특성 [2/2]

- 포인터 연산을 제거
- 자료 손실이 가능한 묵시적인 형 변환 제거
  - 명시적인 cast 연산 필요

```
int     myInt ;
float   myFloat = 3.141592f ;
myInt  = myFloat ;
```

```
int     myInt ;
float   myFloat = 3.141592f ;
myInt  = (int) myFloat ;
```

- **Incompatible type for =. Explicit cast needed to convert float to int.**
  myInt = myFloat;

- 연산자 중복(operator overloading)을 제거
- 메모리 관리
  - malloc()을 제거

특성 2. 객체지향 언어

- 객체지향 언어 특성
  - 자료 추상화(data abstraction)
  - 상속성(inheritance)
  - 다형성(polymorphism)
- 다형성
  - Function overloading → method overloading
  - Operator overloading → 지원하지 않음

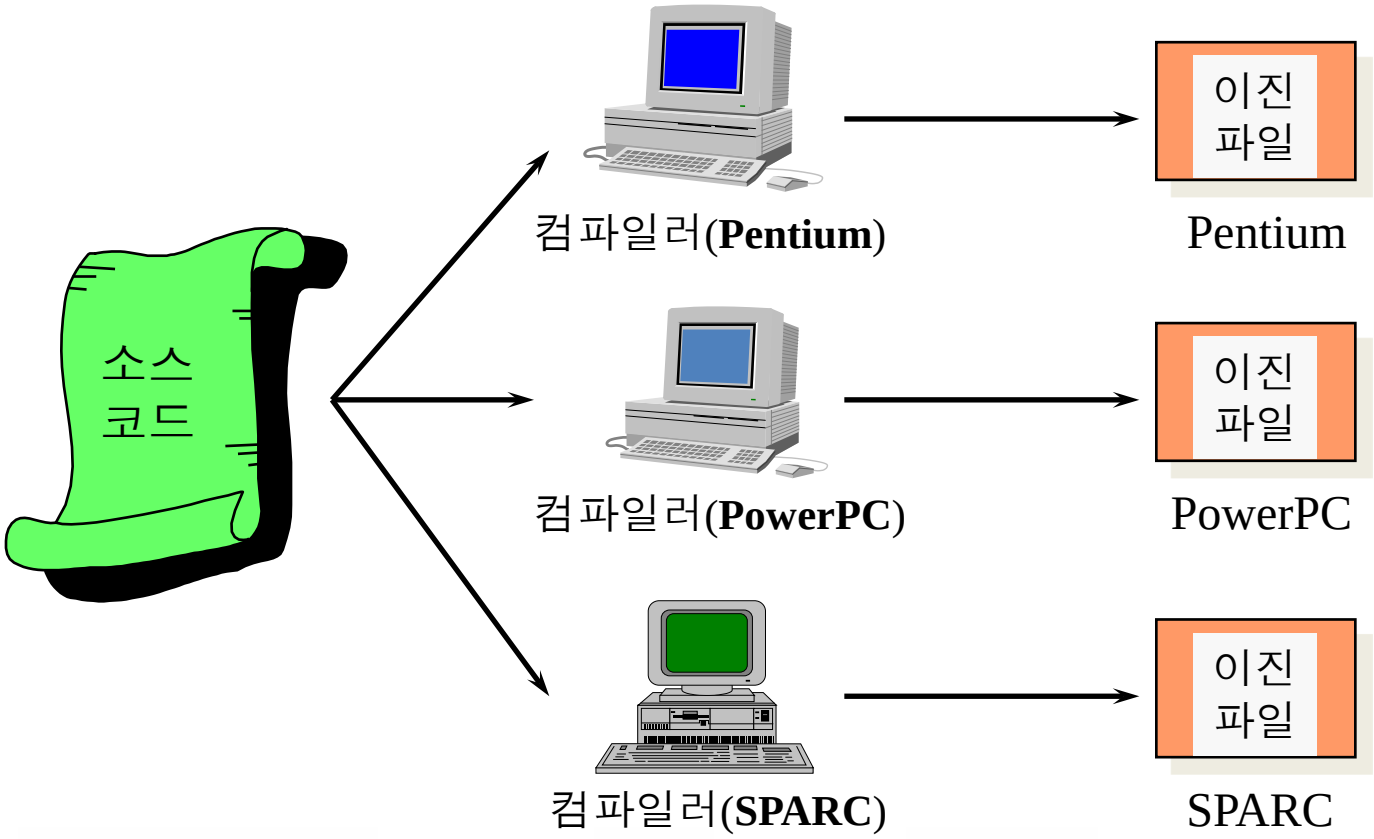
특성 3. 플랫폼에 독립적 [1/3]

- 아키텍처에 독립적
  - 플랫폼 독립적인 중간코드 사용 – 바이트코드(bytecode)
  - 가상기계를 이용하여 실행 – JVM(Java Virtual Machine)

- 기존 개발환경과의 비교
  - 전통적인 컴파일링 시스템
  - 플랫폼에 독립적인 시스템

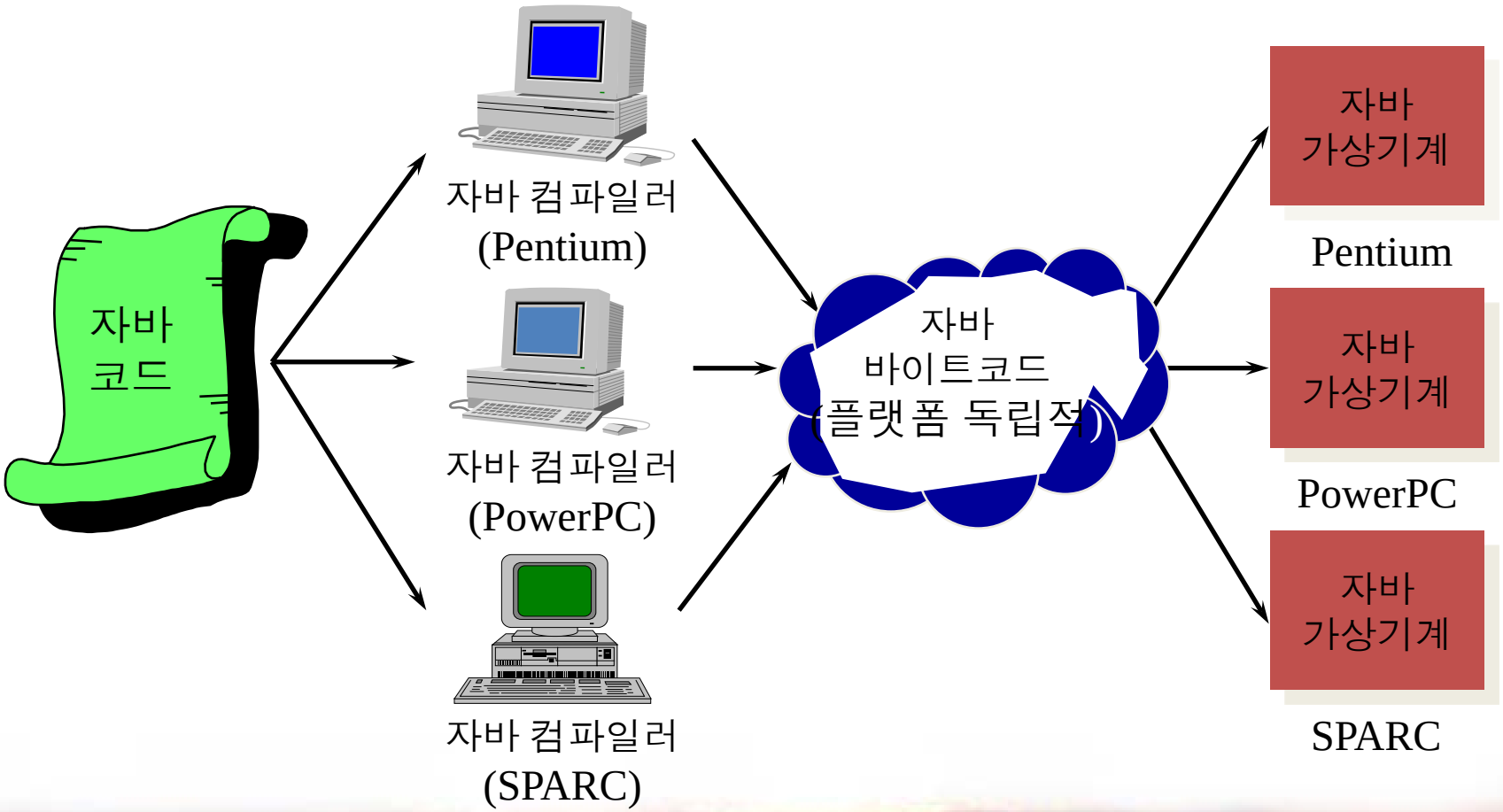
특성 3. 플랫폼에 독립적 [2/3]

- 기존 프로그램 개발 환경



특성 3. 플랫폼에 독립적 [3/3]

- 자바 개발 환경



자바 컴파일 및 실행 과정

HelloWorld.java

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

자바 컴파일러 (javac)

```
C:\JDK\BIN>javac HelloWorld.java
```

HelloWorld.class

자바 가상기계 (java)

```
C:\JDK\BIN>java HelloWorld
```


```
Hello World!
```

자바 가상 기계와 자바 응용프로그램의 실행

자바 프로그래밍

Draw.java
Hello.java
Shape.java

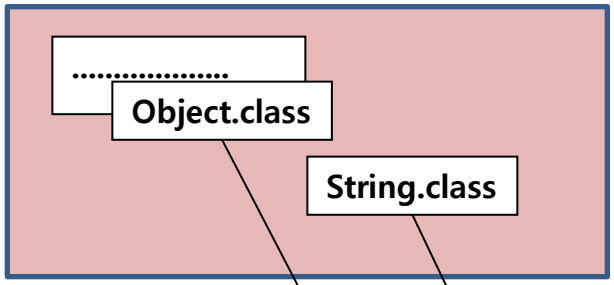
자바 컴파일러

Draw.class
Hello.class
Shape.class

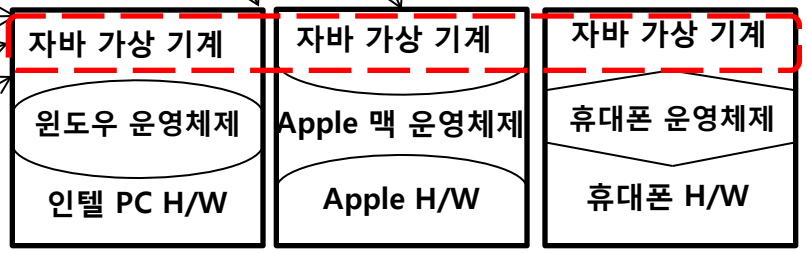
(소스 코드)

(바이트 코드)

실행에 필요한 자바 클래스 라이브러리(JDK APIs)



클래스 로딩



애플릿 만들기

- 간단한 프로그램

```
import java.awt.Graphics ;

public class HelloWorldApplet extends java.applet.Applet {
      public void paint( Graphics  g ) {
            g.drawString( "Hello World!", 5, 25 ) ;
      }
}
```

import java.applet.Applet;

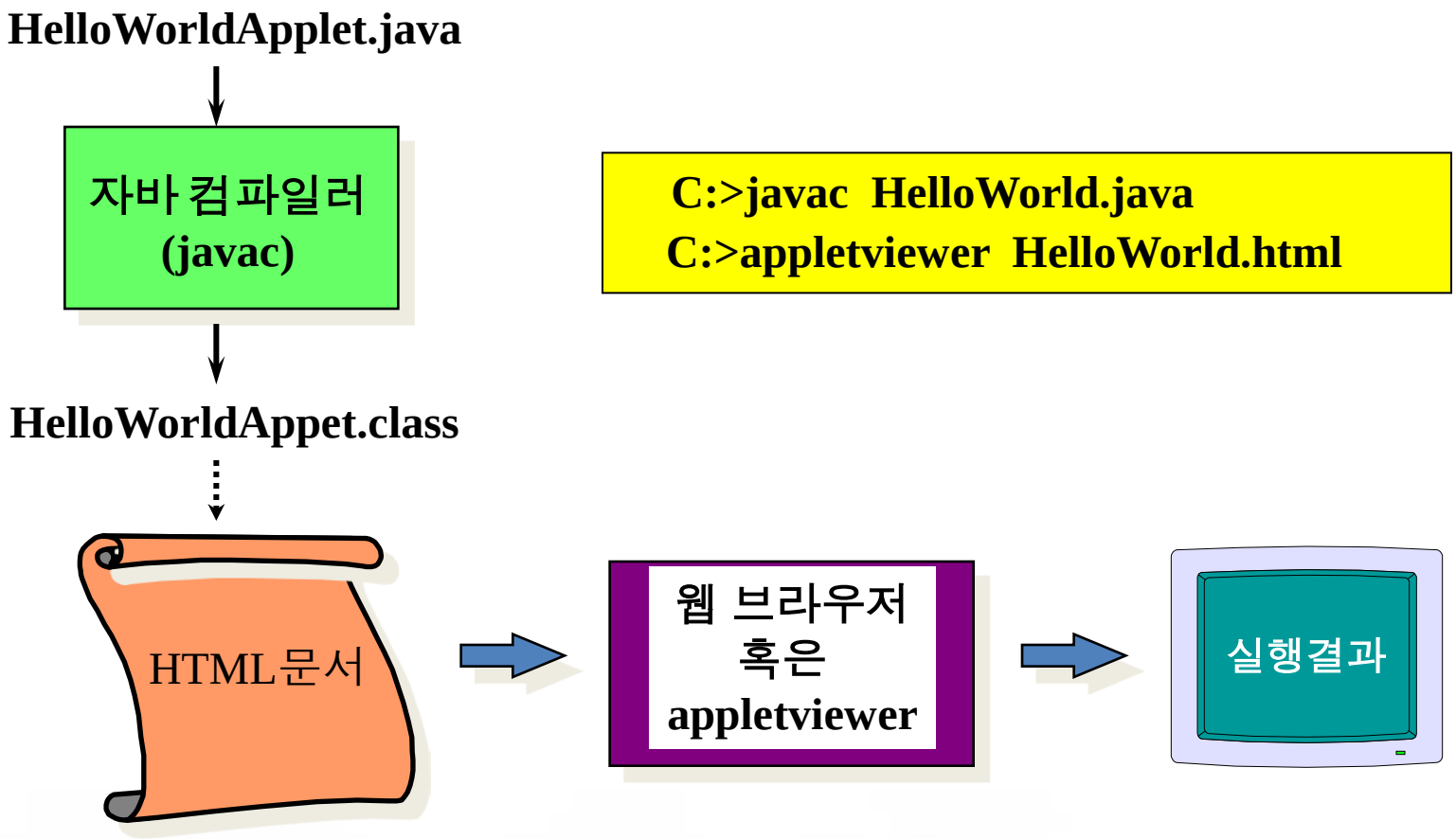
- java.applet.Applet
- paint(Graphics g)
- import 문

애플릿 만들기

- [예제 1.3 – HelloWorldApplet.java]

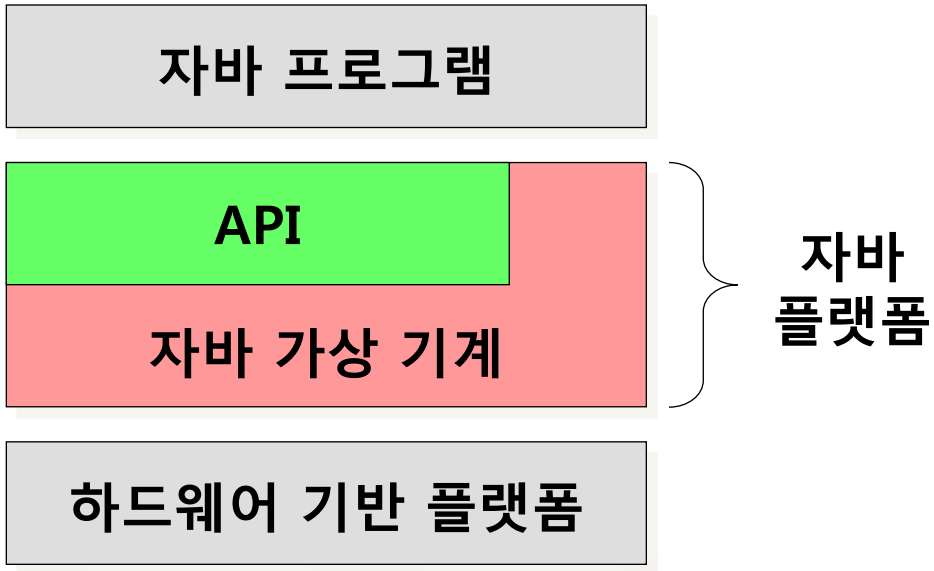
```
<HTML>
  <HEAD>
    <TITLE> HelloWorldApplet </TITLE>
  </HEAD>
  <BODY>
    <APPLET CODE="HelloWorldApplet.class" WIDTH=300 HEIGHT=80>
    </APPLET>
  </BODY>
</HTML>
```

자바 애플릿 실행과정


HelloWorldApplet.java
자바 컴파일러
(javac)
HelloWorldAppet.class
HTML문서
웹 브라우저
혹은
appletviewer
실행결과
C:>javac HelloWorld.java
C:>appletviewer HelloWorld.html

자바 플랫폼

- 자바 플랫폼 : 자바 가상 기계와 API
  - 장점 : 하드웨어 플랫폼 독립성
  - 단점 : 실행 속도의 저하



실행 성능 향상 방법

- Java Chip
  - 바이트코드를 직접 실행하는 프로세서를 적용
- JIT(Just-In Time) Compilation
  - 실행 시간에 필요에 따라 메소드 단위로 바이트코드를 목적코드로 바꾸어 실행하는 방법
- Back-End
  - 바이트코드를 목적코드로 모두 바꾼 후 실행
- Decompilation
  - 바이트코드를 C언어와 같이 효율이 좋은 고급 언어로 역컴파일하여 실행하는 방법

특성 4. 견고하고 보안에 강하다

- 포인터 제거
  - 실행컴퓨터 환경을 변경하는 것을 방지
  - 디버깅이 어려운 run-time 에러 발생 감소
- 엄격한 형 검사
  - Strongly typed language
- 보안
  - Class loader의 검사

특성 5. 동적이고 멀티 스레드를 지원

- 동적 링크
  - 클래스가 실행시간에 동적으로 로드
  - 재컴파일 문제점 해결
- 언어차원에서 멀티스레드를 지원
  - 서로 다른 일을 동시에 처리 가능
  - 하나의 자바 프로그램이 여러 개의 스레드로 구성

자바 개발 환경 [1/2]

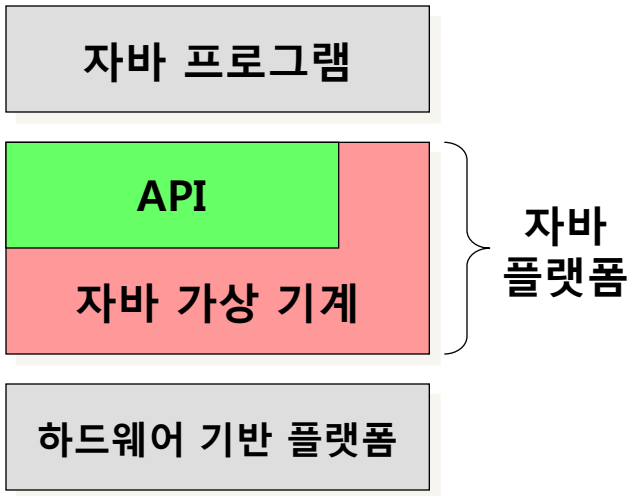
- 프로그래밍 언어 시스템
  - 프로그래밍 환경과 운영환경으로 구성된다.
- 프로그래밍 환경 : 프로그램 개발용 도구
  - 편집기, 디버거
  - 독립된 편집기를 사용하기도 하지만 대부분 IDE에 포함된 편집기를 사용한다.
- 운영 환경 : 프로그램 실행 환경
  - 컴파일러, 실행시간 지원 시스템, 라이브러리 시스템
  - JDK(Java Development Kit)

자바 개발 환경 [2/2]

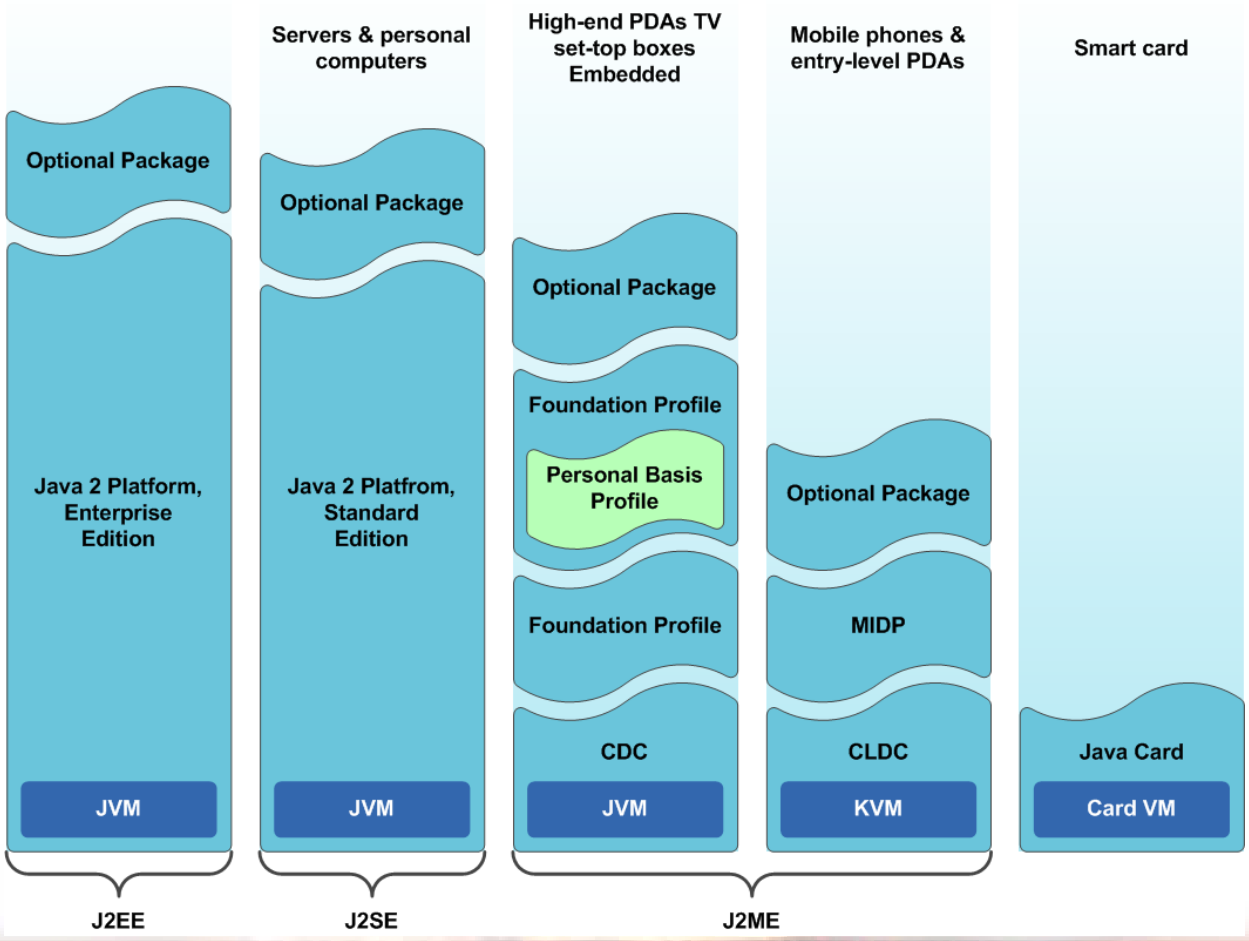
- 통합 개발 환경(IDE: **I**ntegrated **D**evelopment **E**nvironment)
  - 프로그래밍 환경과 운영 환경이 하나로 결합되어 프로그램의 작성부터 실행까지 대화식으로 수행 가능한 개발 환경
  - 이클립스, 넷빈즈
- 이클립스(Eclipse)
  - 이클립스 재단에서 개발
  - 플러그인(plug-in) 개념을 가진 공개 통합 개발 환경
  - http://www.eclipse.org
- 넷빈즈(NetBeans)
  - Sun사가 제공하는 무료 통합 개발 환경, JDK 필요
  - http://www.netbeans.org

자바의 종류 [1/2]

- 자바 플랫폼
  - 자바 가상 기계와 API
  - 하드웨어의 종류에 따라 다양한 자바 플랫폼 존재
    - 자바 표준 에디션 (Java SE: Java Standard Edition)
    - 자바 엔터프라이즈 에디션 (Java EE: Java Enterprise Edition)
    - 자바 모바일 에디션 (Java ME: Java Mobile Edition)
    - 자바 카드(Java card)



자바의 종류 [2/2]

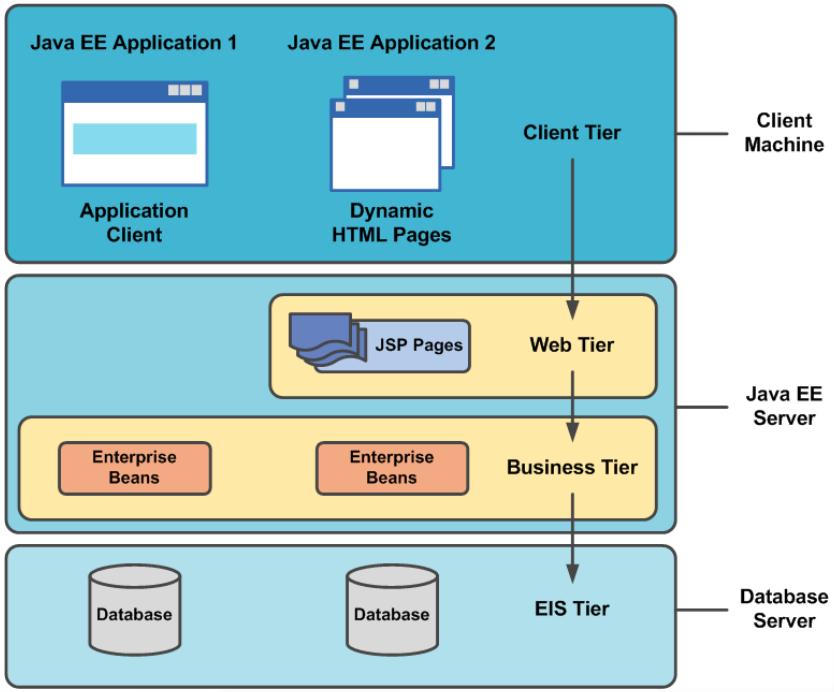

Servers & personal computers
High-end PDAs TV set-top boxes Embedded
Mobile phones & entry-level PDAs
Smart card
Optional Package
Optional Package
Optional Package
Optional Package
Java 2 Platform, Enterprise Edition
Java 2 Platfrom, Standard Edition
Foundation Profile
Personal Basis Profile
Foundation Profile
CDC
MIDP
CLDC
Java Card
JVM
JVM
JVM
KVM
Card VM
J2EE
J2SE
J2ME

자바 표준 에디션

- Java SE(Standard Edition)
  - 주로 데스크탑용 프로그램에 사용
  - 개발 : JDK, 이클립스, 넷빈즈 등
  - 실행 : JRE(Java Runtime Environment) 설치 필요
- 주요 응용 분야
  - 콘솔 프로그램 : 문자기반의 콘솔을 통해 입출력 수행
  - 그래픽 프로그램 : 그래픽 사용자 인터페이스(GUI : Graphic User Interface)를 통해 입출력 수행
  - 애플릿 : 웹브라우저에서 실행되는 간단한 자바 프로그램

자바 엔터프라이즈 에디션

- Java EE(Enterprise Edition)
  - 기업용 프로그램 작성을 위한 분산 객체, 트랜잭션, 높은 이식성 제공
  - 개발 : 별도의 Java EE SDK 사용
  - 실행: 여러 종류의 애플리케이션의 조합으로 동작



자바 마이크로 에디션

- Java ME(Micro Edition)
  - 내장형기기(embedded device)를 위한 자바 플랫폼
    - 휴대폰, PDA, 셋탑박스 등 성능이 낮고, 메모리가 작은 기기
    - 다양한 CPU와 운영체제가 사용되어 플랫폼 독립성이 중요
  - 휴대폰이나 PDA에서 실행되는 간단한 게임, TV 프로그램 가이드, 쇼핑 프로그램 등

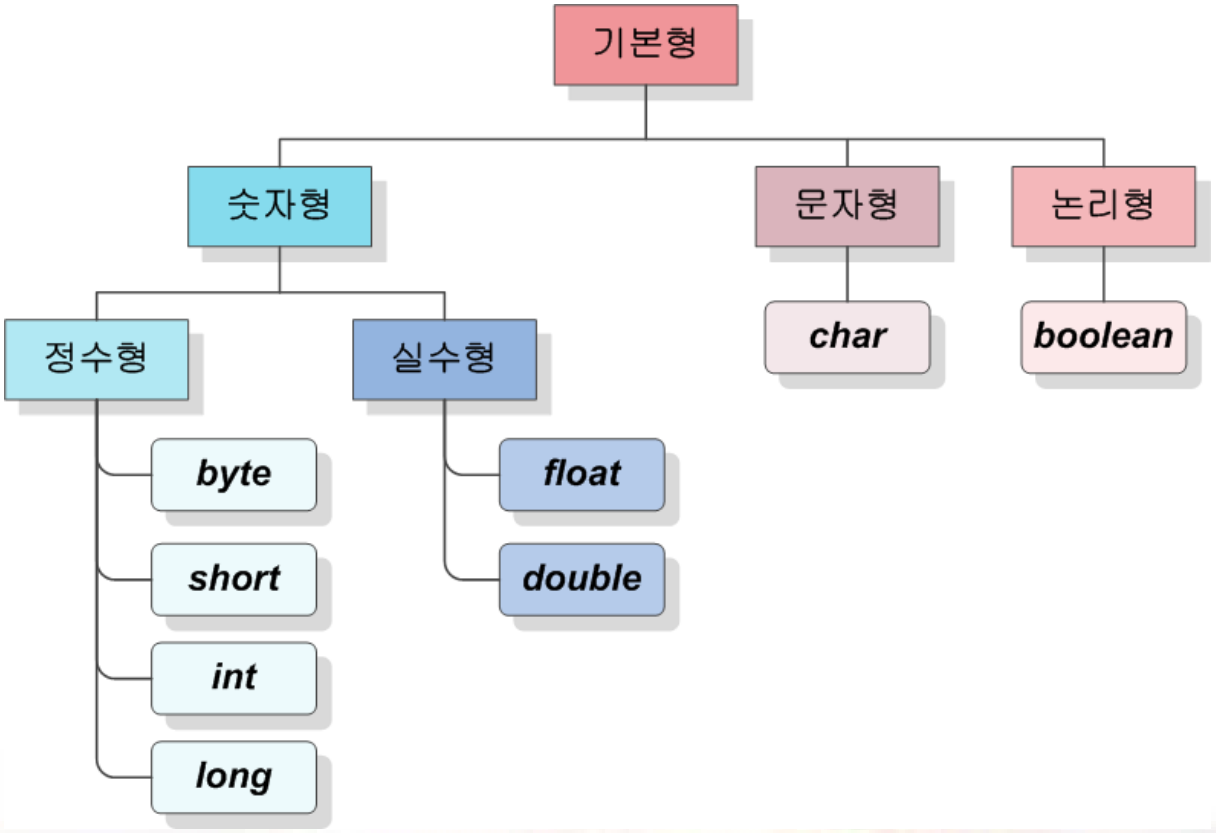
- Configuration과 Profile
  - Configuration : 기기의 성능에 따라 구분한 API 명세
  - Profile : 요구되는 기능에 따라 특화된 API 모임

자바의 기본 패키지

- java.lang
  - 자바의 기능을 확장해 주는 기본적인 클래스 : Object, String 등
  - 마치 default library처럼 자바 컴파일러에 의해 자동으로 import
- java.io
  - 스트림 입출력, 파일 입출력에 관련된 클래스 제공
  - java.net 패키지는 소켓, 텔넷 인터페이스, URL 클래스 제공
- java.util
  - 프로그래머에게 유용한 벡터, 스택 , 해쉬테이블 등...
- java.awt (Abstract Window Toolkit)
  - 플랫폼 독립적인 GUI 개발 도구
  - java.swing은 자바로 구현되어 어느 시스템에서도 동일하게 보임

자바의 기본 특징 [1/7]

- 기본형(primitive type)



자바의 기본 특징 [2/7]

- 기본형에 대한 예제

[예제 1.4 – Primitives.java]

```
public class Primitives {
    public static void main(String[] args) {
        boolean b;
        int i = Integer.MAX_VALUE;
        double d = Double.MIN_VALUE;

        b = ( i != 0 );
        System.out.println("boolean b = " + b);
        System.out.println("max value of integer  = " + i);
        System.out.println("min value of double   = " + d);
    }
}
```

실행 결과 :

```
boolean b = true
max value of integer  = 2147483647
min value of double   = 4.9E-324
```

자바의 기본 특징 [3/7]

- 연산자
  - 부호없는 정수형이 존재하지 않음
  - '>>>' 연산자 --- unsigned right shift
  - '+' 연산자 --- string concatenation

자바의 기본 특징 [3/7]

[예제 1.5 – **Operators.java**]

```
public class Operators {
  public static void main(String[] args) {
      int i = -1;
      int j =  7;

      int neg = i >>> 1;
      int pos = j >>> 2;

      System.out.println("-1 >>> 1 = " + neg);
      System.out.println(" 7 >>> 2 = " + pos);
   }
}
```

실행결과 :

-1 >>> 1 = 2147483647

7 >>> 2 = 1

자바의 기본 특징 [4/7]

- 배열 - 객체(object)로 처리
  - 배열 변수 선언

```
 int[]      vector;
short      matrix[][];  // short[][]  matrix;
Object[]   myArray1, myArray2;
```

  - 배열 객체 생성

```
 vector = new int[5];
matrix = new short[10][100];
myArray1 = new Point[3];
```

  - 배열에 값 저장

```
 int  a[] = new int[50];
for(int i=0; i < a.length; i++) a[i] = i;
```

자바의 기본 특징 [5/7]

- 스트링 - 문자의 배열이 아닌 객체로 처리
  - 스트링 객체 생성

```
String hello1 = "Hello world!";
String hello2 = new String("Hello world!");
```

  - '+' 연산자(concatenation)

```
String hello = "Hello " + "world!";
```

자바의 기본 특징 [6/7]

- 개선된 for 문

```
String[] color = { "red", "green", "blue" };

for (String s: color) {
    System.out.println(s);
}
```

- 다중 break 문

```
test: for (int i=0; i<max1; i++)
          for (int j=1; j<max2; j++)
              if (matrix[i][j] == 0)
                  break test;
```

자바의 기본 특징 [7/7]

- 메모리 관리와 가비지 수집
  - 힙(heap)에 대한 메모리 관리
  - 자동적인 메모리 수집
  - ➔ 프로그래머의 부담을 줄여 준다.

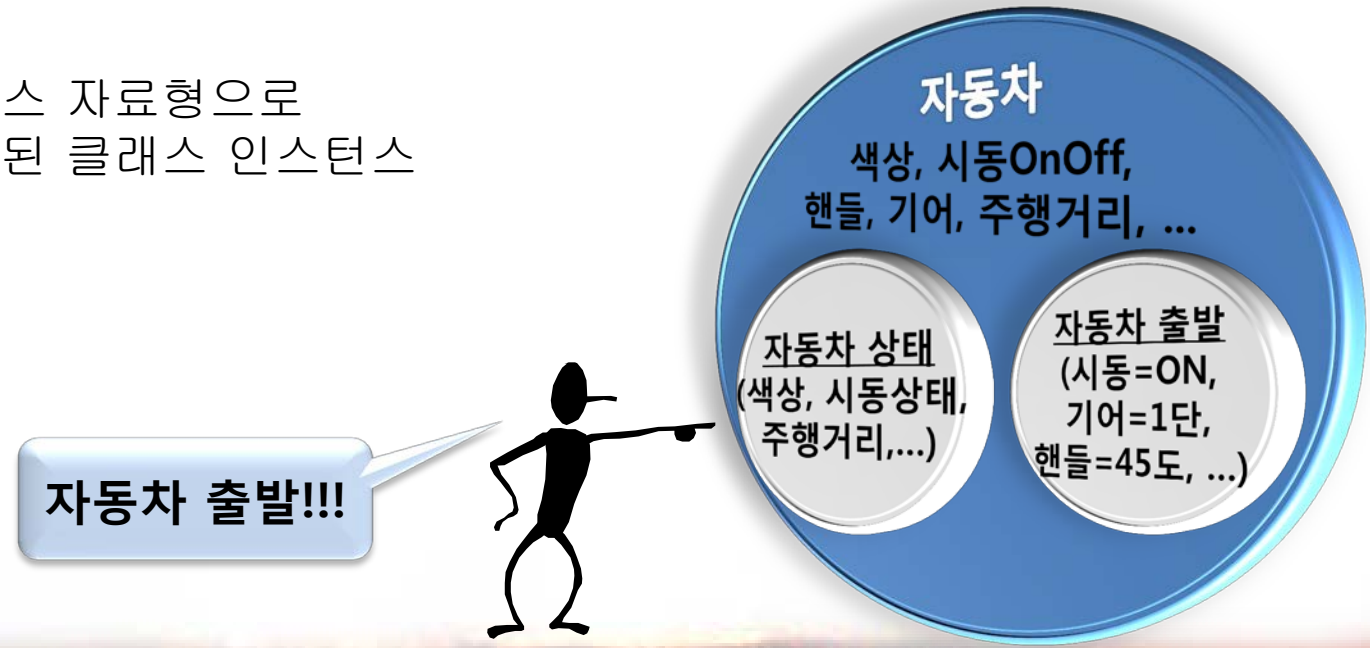
- Java와 C++의 비교
  - Allocation(constructor) : C++(o), Java(o)
  - Deallocation(destructor) : C++(o), Java(x)

자바의 주요 특성

1. 클래스(Class)
2. 제네릭(Generics)
3. 예외처리(Exception)
4. 스레드(Thread)

특성 1. 클래스 [1/3]

- 클래스 : 객체를 표현하는 자료형
- 클래스의 구성
  - 필드(field) : 변수, 상수
  - 메소드(method)
- 객체
  - 클래스 자료형으로 선언된 클래스 인스턴스



특성 1. 클래스 [2/3]

- 클래스 설계

```
class CoffeeMaker {
    String   color;
    boolean  onState;

    void startCoffeeMaker() {
        if (onState == true)
            System.out.println("The CoffeMaker is already
on");
        else {
            onState = true;
            System.out.println("The CoffeMaker is now on");
        }
    }
}
```

특성 1. 클래스 [3/3]

- 클래스 호출

```
public class Coffee {
    public static void main(String args[]) {
        CoffeeMaker cm = new CoffeeMaker();
        cm.onState = true;
        cm.start();
    }
}
```

특성 2. 제네릭

- 범용 클래스, 포괄 클래스
  - 자료형을 매개변수로 하여 상이한 자료형에 대해 동일한 연산을 정의
  - 범용 클래스 정의

```
class Stack<StackType> {
    private Vector<StackType>[] stack = new
Vector<StackType>();
    // ...
    public void Push(StackType element) { /* ... */ }
    public StackType Pop() { /* ... */ }
}
```

  - 객체 생성

```
    Stack<Integer> stk1 = new Stack<Integer>();          // 정
수형 스택
    Stack<Double> stk2 = new Stack<Double>();          // 실
수형 스택
```

특성 3. 예외처리 [1/2]

- 예외 : run-time error(cf. compile error)
- 예외 처리
  - 자바는 예외를 체계적으로 처리할 수 있는 기능을 언어 수준에서 지원
  - 예외 발생 ➔ 작성된 예외처리기를 통해 처리
- 예외처리를 위한 구문

```
try {
    // ...                                  "try 블록"
} catch(ExceptionType identifier) {
    // ...                                  "catch 블록"
} finally {
    // ...                                  "finally 블록"
}
```

특성 3. 예외처리 [2/2]

- 예외처리 예제

[예제 1.9 – ExceptionExample.java]

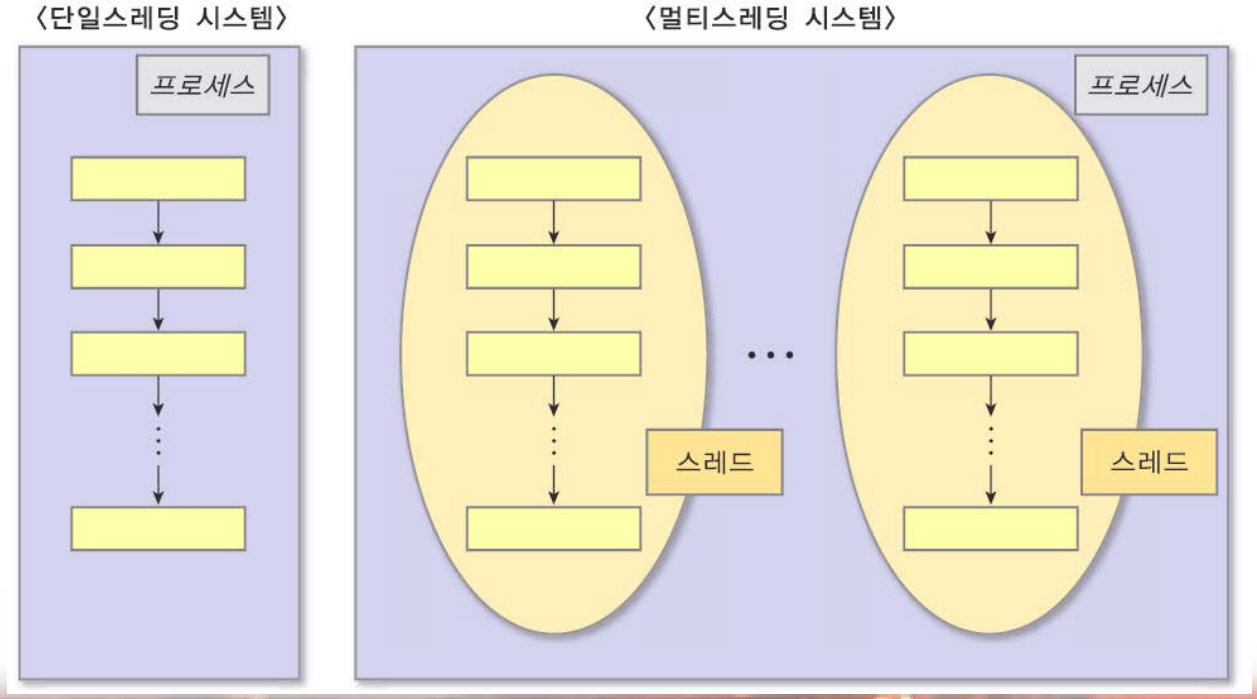
```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            System.out.println("Exception throwing...");
            throw new Exception();
        } catch(Exception e) {
            System.out.println("Caught Exception");
        } finally {
            System.out.println("In the finally statement...");
        }
    }
}
```

실행 결과 :

```
Exception throwing...
Caught Exception
In the finally statement...
```

특성 4. 스레드(Thread) [1/2]

- 시작, 실행, 종료의 순서를 가진 제어의 흐름
- 멀티스레딩 시스템
  - 공유 힙(heap), 공유 데이터, 코드 공유



특성 4. 스레드(Thread) [2/2]

[예제 1.10 - ThreadExample.java]

```
class ExtnOfThread extends Thread {
    public void run() {
        System.out.println("Extension of Thread, 1");
        try {
            sleep(1000);
        } catch (InterruptedException ie) { }
        System.out.println("Extension of Thread, 2");
    }
}
class ImplOfRunnable implements Runnable {
    public void run() { System.out.println("Implementation of Runnable"); }
}
public class ThreadExample {
    public static void main(String[] args) {
        ExtnOfThread t1 = new ExtnOfThread();
        t1.start();
        Thread t2 = new Thread (new ImplOfRunnable());
        t2.start();
    }
}
```

실행 결과 :

```
Extension of Thread, 1
Implementation of Runnable
Extension of Thread, 2
```