Java & Servlet/JSP 프로그래밍

Part 1.

강경미 (carami@nate.com)
(주)써니베일

목차

객체지향의 기본개념

1. 도서 대여점에서 다음과 같은 일은 누가 할까요?

- 책을 관리하는 것은 누구인가요?

- 고객을 관리하는 것은 누구인가요?

- 돈을 관리하는 것은 누구인가요

2. 야수가 사는 성에 등장하는 인물은?

❑ “미녀와 야수”라는 애니메이션을 아시나요? 이 애니메이션에서 야수가 사는 성에 등장하는 인물 중에서 알고 있는 인물들을 말하시오.

주전자, 촛대,
시계, 책장 등

물체들이 인물인가요?

3. 도서 대여점도 야수의 성처럼 마법에 걸려 있다고 생각합시다.

- 도서 대여점도 야수의 성처럼 마법에 걸려 있다고 생각해 봅시다. 이 경우 어떤 물체들이 살아서 움직이고 있을까요?

- 만약 “책장”이 살아서 움직이고 있다고 생각해 봅시다. 책장에게 이런 질문을 할 수 있을 것입니다.

  - 가지고 있는 책은 몇 권 이니?
  - “도가니” 라는 소설을 혹시 가지고 있니?
  - “이문열”이 쓴 책은 어떤 것을 가지고 있니 ?

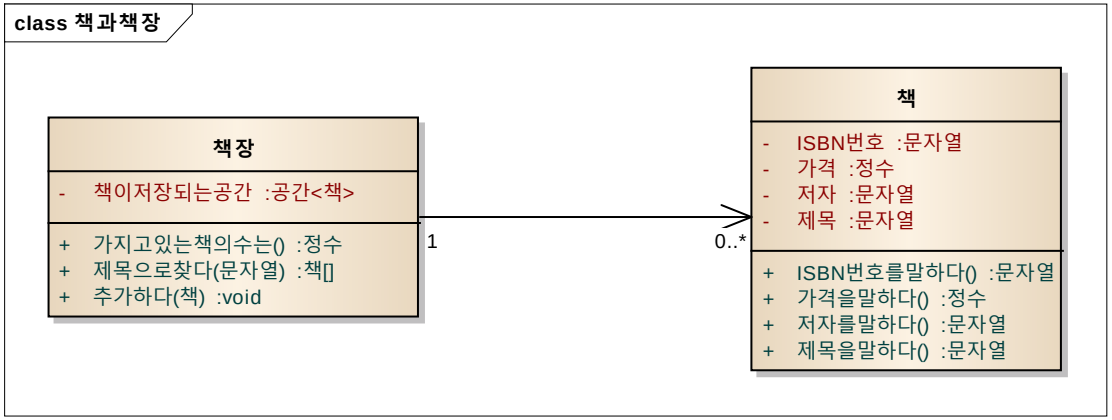
- 책을 가지고 있는 것은 책장입니다. 야수의 성처럼 마법에 걸려 있다면, 책을 관리하는 것이 무엇인가? 라고 물어본다면 “책장”이라고 말할 수 있습니다.

4. 그렇다면 도서대여점에 대하여 다시 질문 드리도록 하겠습니다.

- 도서 대여점에서 책을 관리하는 것이 “책장”이라고 하였습니다. 그렇다면 돈을 관리하는 것은 누구일까요?

- 조금은 어려운 질문 일수도 있는데요. 그렇다면 “고객”을 관리하는 것은 누구일까요?

5. has a 관계



- 위의 그림을 말로써 표현하자면 다음과 같습니다.
  - 책장은 책을 가질 수 있다.
  - 책장은 책을 저장하는 공간을 가지고 있다.
  - 책은 ISBN번호, 가격, 저자, 제목을 가지고 있다.
  - 책장과 책은 자신이 가지고 있는 속성을 이용하는 기능을 가지고 있다.
- **가지는 관계 : has a 관계**

6. 고객과 고객장부도 그려보세요.

❑ 앞의 책과 책장을 참고로 하여 고객과 고객 장부도 아래에 그려 보세요.

7. 도서 대여점의 고객 vs 신발 가게의 고객

- 도서 대여점의 고객에게 신발 사이즈를 물어 본다면?
- 도서 대여점의 고객에게 몸무게를 물어 본다면?
- 신발 가게의 고객에게 신발 사이즈를 물어 본다면?

- 도서 대여점의 고객은 도서 대여점 주인 입장으로 봤을 때, 책을 빌려 간 후 회수해야 하기 때문에 이름과 연락처가 중요하다. 하지만, 신발 사이즈와 몸무게는 중요하지 않다. 신발 사이즈와 몸무게를 물어 본다면 고객은 언짢을 수 있을 것이다.
- 하지만, 신발 가게에서 신발 사이즈를 물어 본다면 신발 가게의 고객은 당연히 신발 사이즈를 말할 것이다.

- 중요한 것은 남기고, 불필요한 것을 없애는 것을 객체 지향에서는 “추상화” 라고 말한다.

8. 모든 것이 중요하지는 않다.

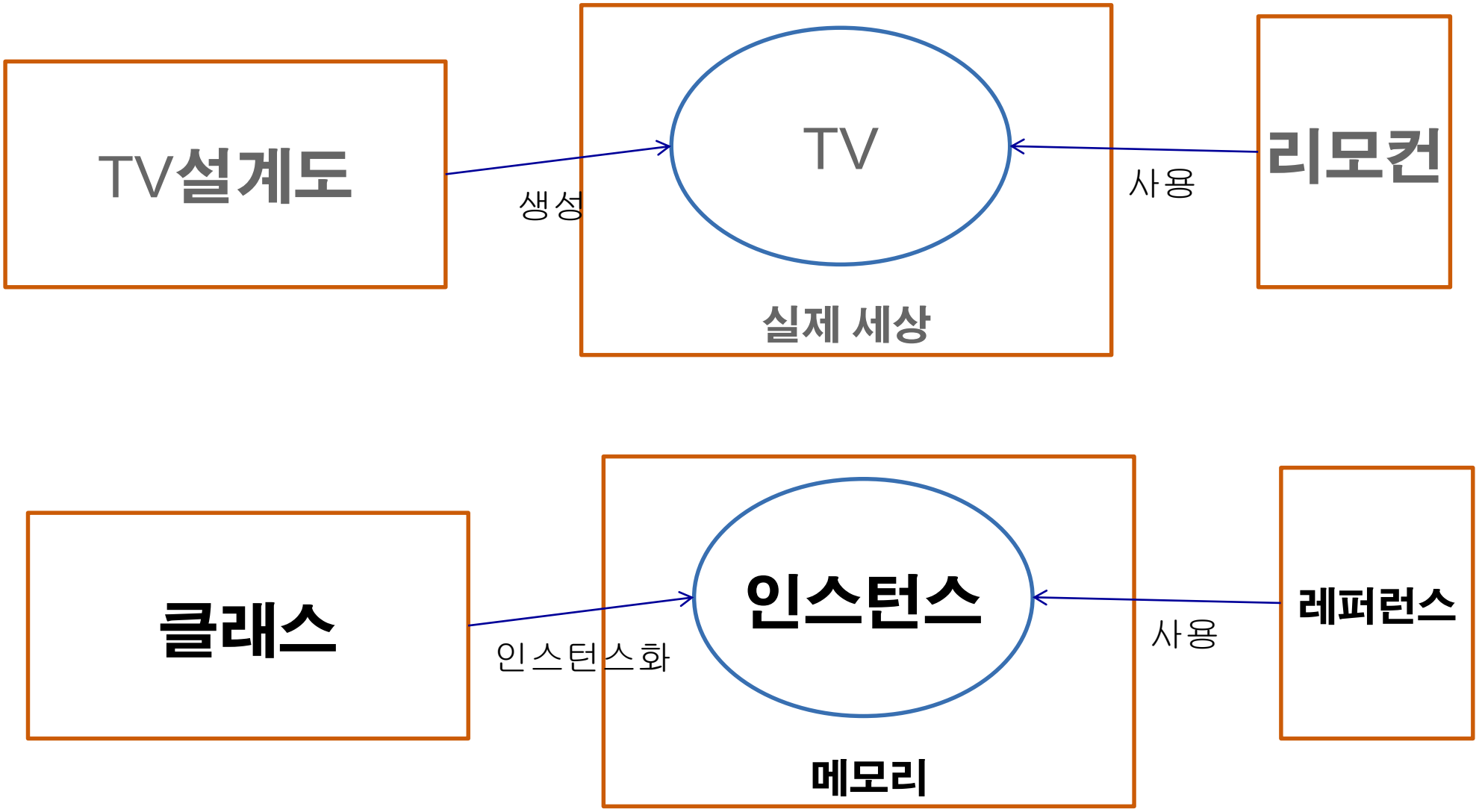
- 도서 대여점에서의 책장은 색깔이 중요하지 않지만, 가구점에서는 책장의 색깔이 중요하다.
- 어떤 관점으로 바라보느냐에 따라서, 객체가 가지는 속성 중 중요한 것과 중요하지 않은 것들로 나뉘게 된다.
- 객체 지향에서 중요한 것은 어떤 관점으로 객체를 바라보느냐? 하는 부분이 있다.
- 자동차를 보더라도, 자동차를 판매하는 사람, 자동차 설계자, 자동차 디자이너, 자동차 회사 사장 등 자동차를 어떻게 바라보느냐에 따라서 중요하고, 중요하지 않은 부분이 틀려질 것이다.
- 객체 지향 프로그래밍은 프로그래밍의 관점에서 객체를 바라본다. 그리고, 객체가 가지는 속성과 기능을 알맞게 표현하게 된다. 이런 이유로 모든 객체는 재사용되기가 어려울 수가 있다. 특정 관점으로 추상화된 객체는 다른 관점에서는 사용되기 어려울 것이기 때문이다.

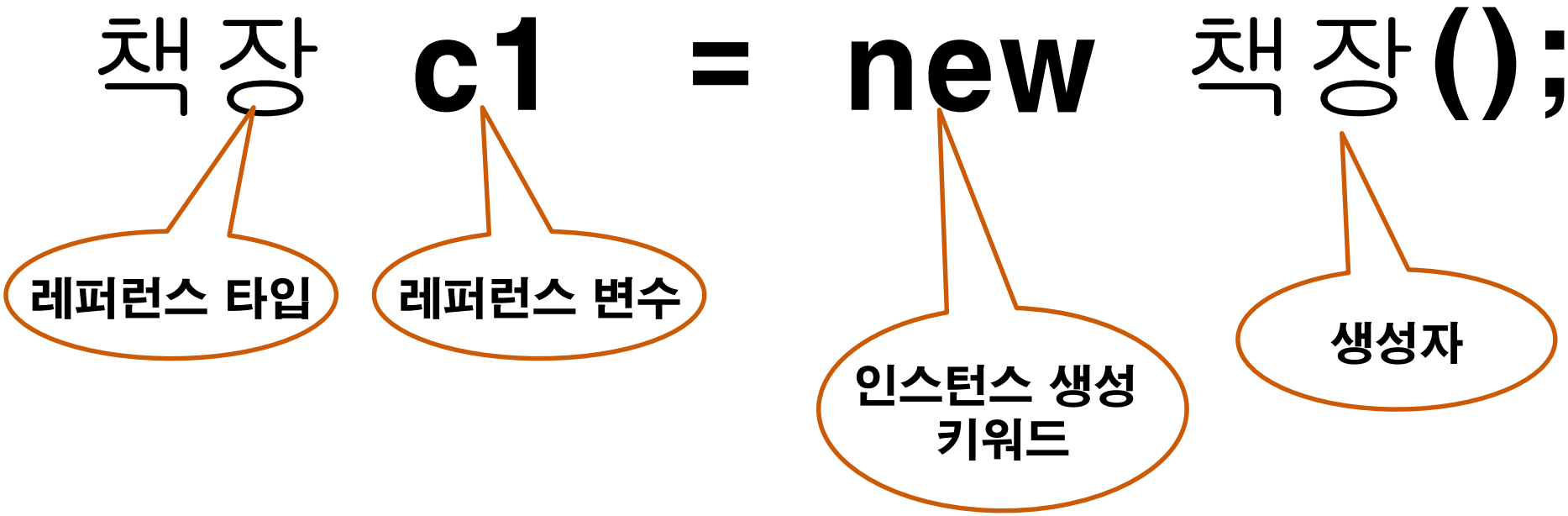
10. 객체 지향이 추구하는 것?

- 현실을 그대로 옮긴다.
- 재사용한다.

- 객체들이 가지고 있는 속성과 기능들 중에서 중요한 것들만 남기고 불필요한 것은 없애게 된다. 이러한 과정을 “추상화 한다.” 라고 말한다.
- 추상화 과정을 통하여 정의된 객체는 기본적으로 재사용 이라는 목적을 가진다. 예를 들어서 도서 대여점의 책장은 도서관에서도 사용될 수가 있다. 하지만, 가구점에서는 해당 책장을 이용하기가 어렵다. 도서 대여점이나 도서관이나 책의 보관과 관리라는 목적으로 책장이 사용 되지만, 가구점에서는 책장이 어떻게 사용되는 것이 중요한 것이 아니기 때문이다. 이런 이유로 모든 객체는 재사용되기 어려울 수 있다.

11. 클래스, 인스턴스, 레퍼런스





책장이라는 인스턴스를 만들고, 이를 책장 타입의 레퍼런스 변수 **c1**이 해당 인스턴스를 레퍼런스(참조)한다.

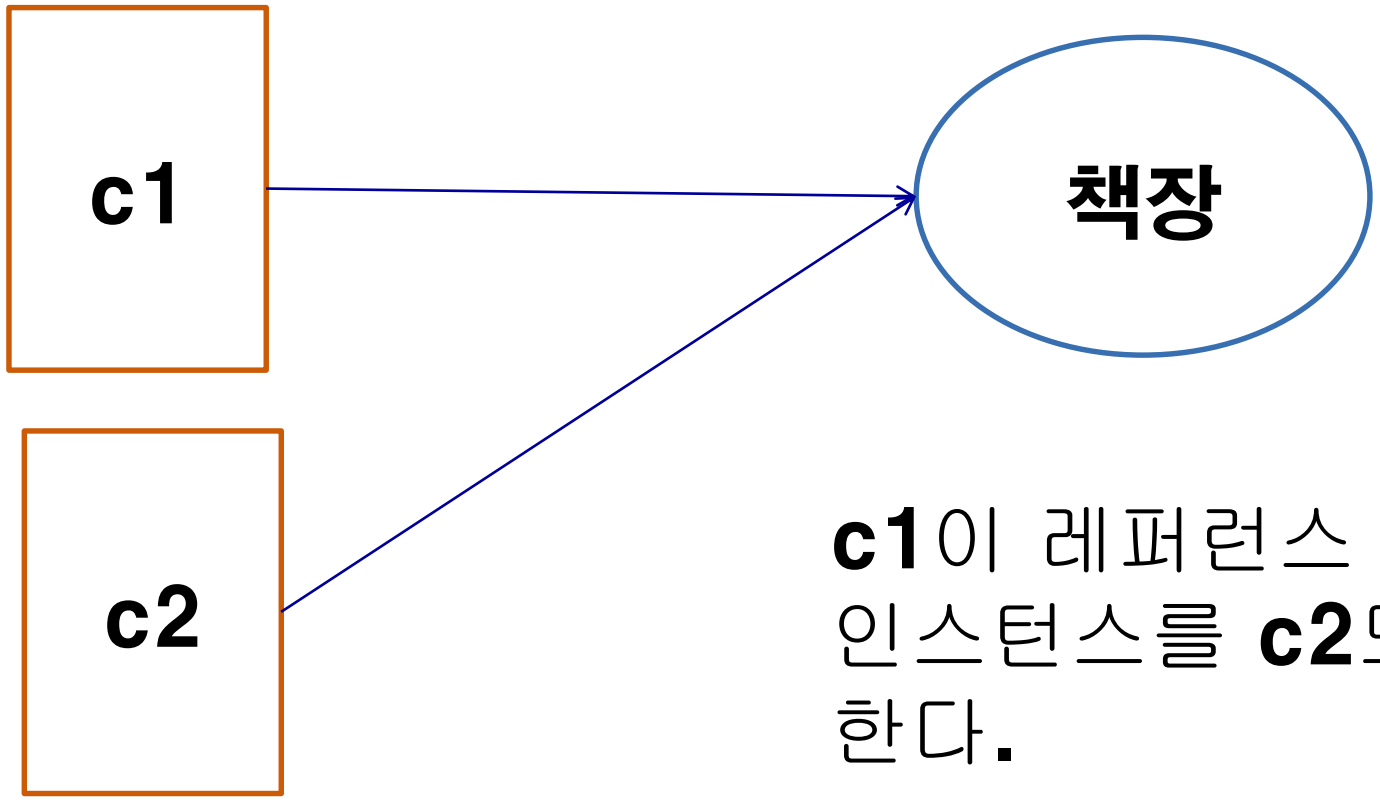
13. 다음과 같은 문장이 10번 실행되면 인스턴스는 몇 개 생성되는가?

- 책장 c1 = new 책장();
- 책장 c2 = new 책장();
- 책장 c3 = new 책장();
- 책장 c4 = new 책장();
- 책장 c5 = new 책장();
- 책장 c6 = new 책장();
- 책장 c7 = new 책장();
- 책장 c8 = new 책장();
- 책장 c9 = new 책장();
- 책장 c10 = new 책장();

10개

14. 다음과 같이 실행되면 인스턴스는 몇 개 생성되는가?

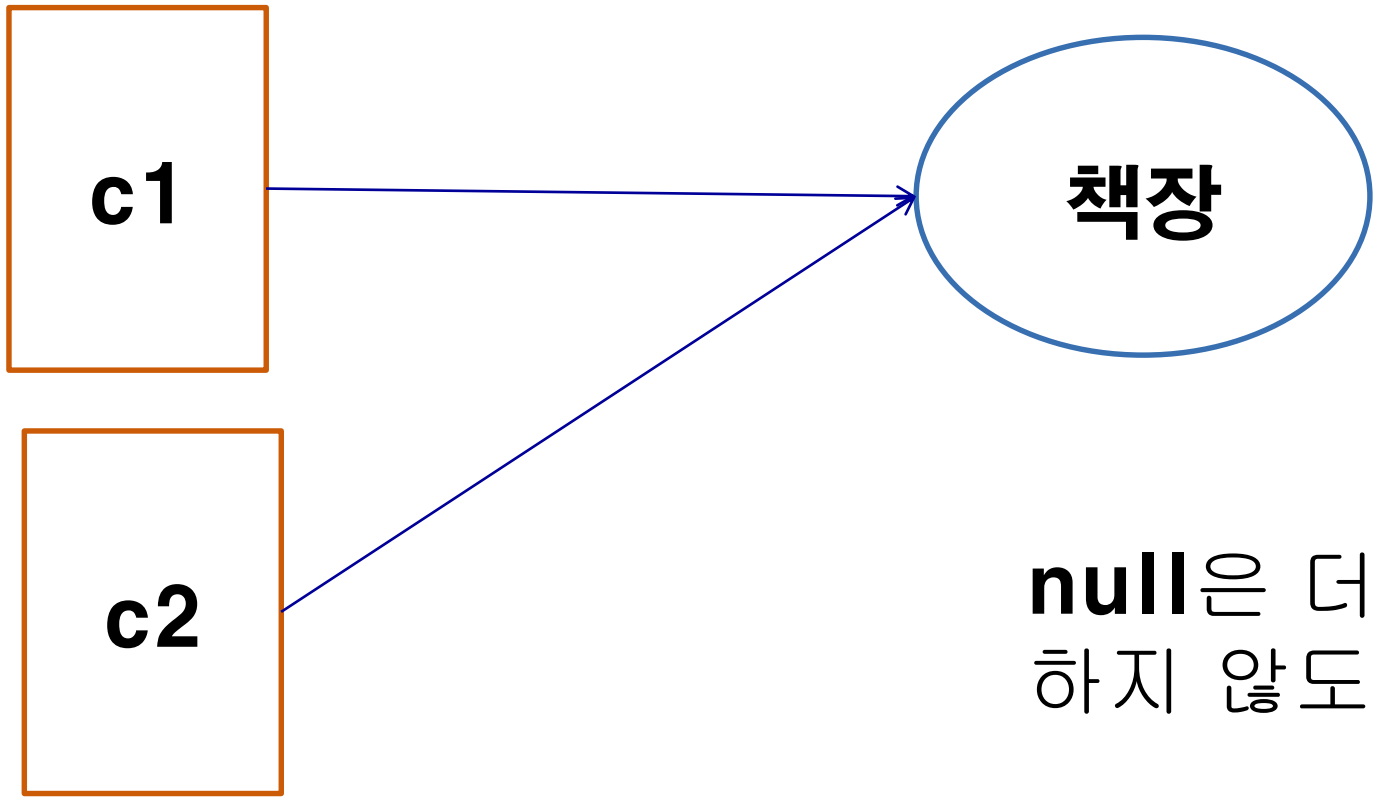
- 책장 c1 = new 책장();
- 책장 c2 = c1;



c1이 레퍼런스 하는 책장 인스턴스를 **c2**도 레퍼런스 한다.

15. 다음과 같이 실행될 경우 레러펀스는 어떤 인스턴스를 가리키게 될까?

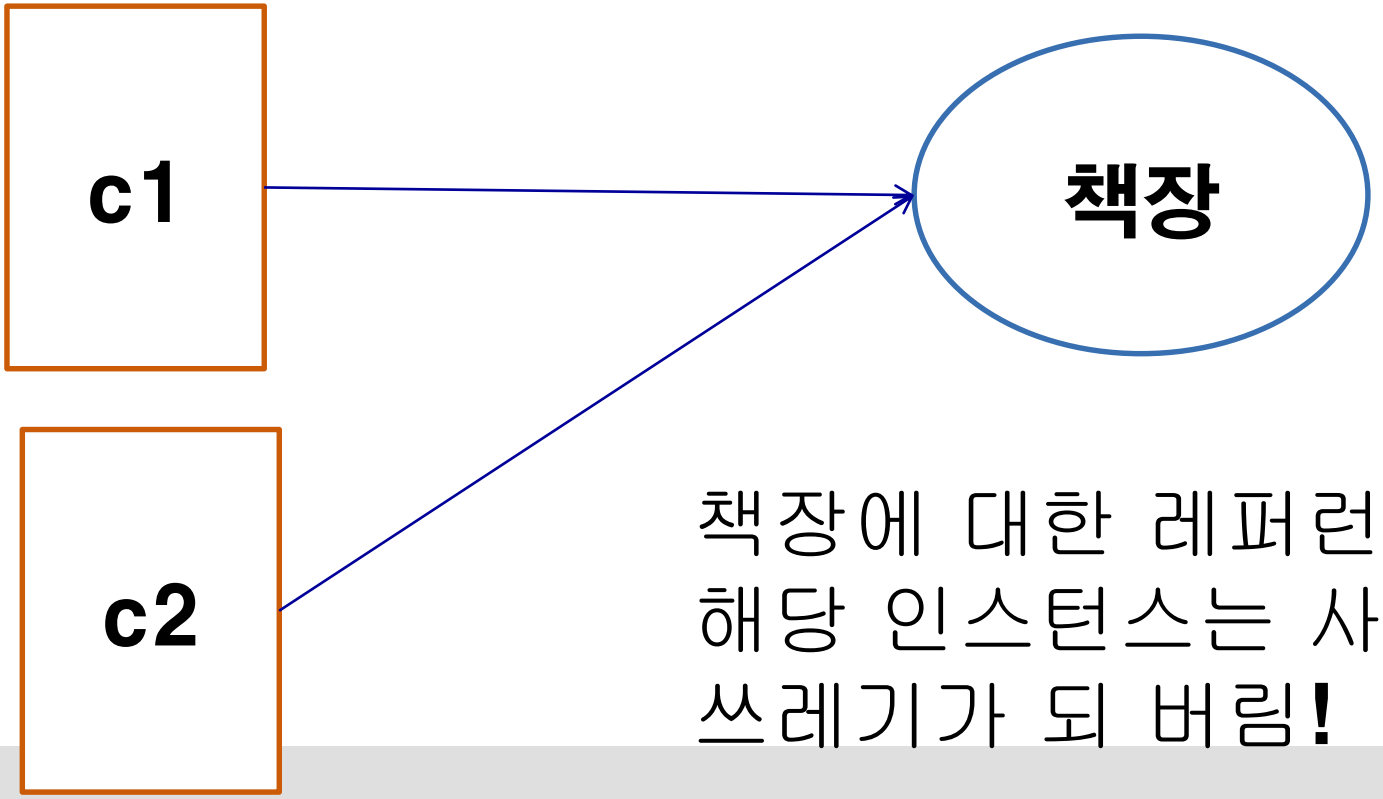
- 책장 c1 = new 책장();
- 책장 c2 = c1;
- c1 = null;



null은 더이상 레퍼런스 하지 않도록 한다.

15. 다음과 같이 실행될 경우 레러펀스는 어떤 인스턴스를 가리키게 될까?

- 책장 c1 = new 책장();
- 책장 c2 = c1;
- c1 = null;
- c2 = null;



책장에 대한 레퍼런스가 없다면,
해당 인스턴스는 사용할 수 없다.
쓰레기가 되 버림!

16. 자동차를 그려 보시오.

❑ 아래에 자동차를 그리세요.

17. 옆자리의 친구에게 다음과 같은 질문을 하세요.

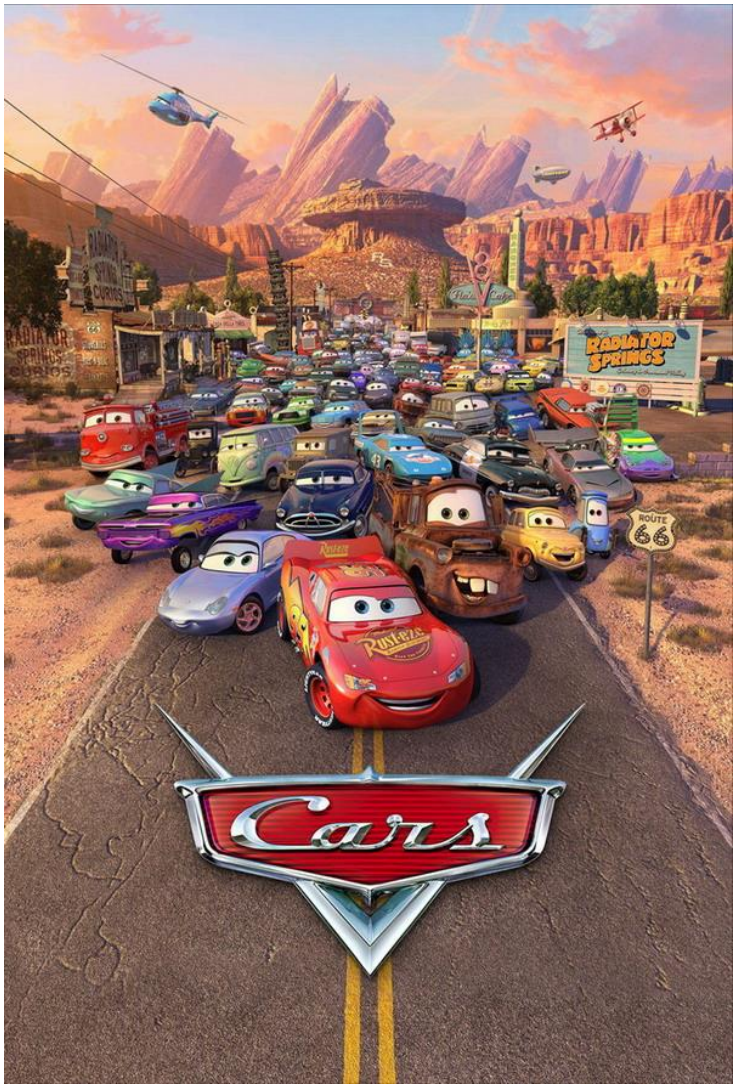
- 저녁에 먹고 싶은 음식의 이름을 구체적으로 말하시오.

- 포유류 중에서 좋아하는 포유류의 이름을 구체적으로 말하시오.

19. 자동차?

자동차(自動車)는 자체 엔진에서 동력을 생산해 바퀴에 전달하여 도로에서 승객이나 화물을 운반하는 교통 수단이다.

- 위키피디아

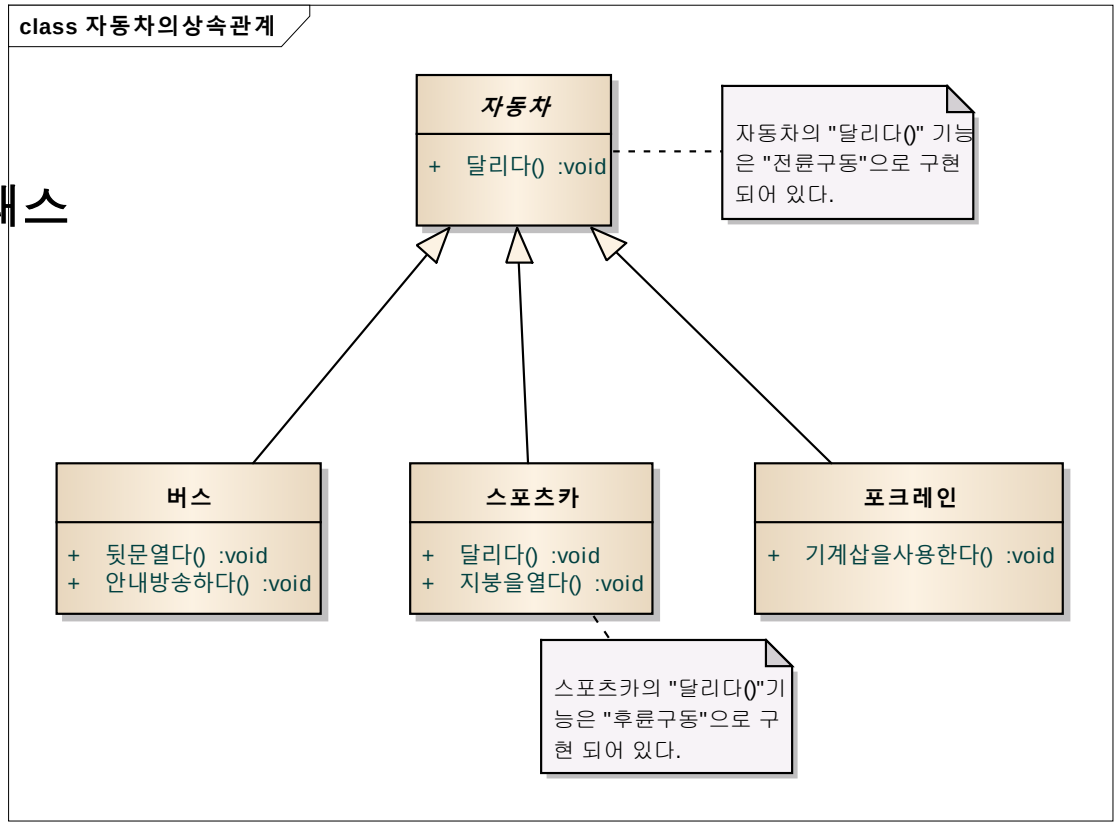


20. 포유류,음식,자동차의 공통점

- 포유류, 음식, 자동차의 공통점은 무엇이라고 생각하는가?
- 포유류, 음식, 자동차는 딱 하나의 물체를 지칭하는 말이 아니다.
- 포유류, 음식, 자동차는 실제로 존재하는 물체를 말할까? 아니면 공통점이 있는 여러 물체들을 하나로 부르기 위해서 존재하는 단어일까?
- 포유류, 음식, 자동차는 “인스턴스”로 존재할 수 있을까?
- 공통점이 있는 여러 가지 물체들을 하나의 이름으로 부르는 것을 “일반화” 라고 한다.
- 이름은 존재하지만, 인스턴스가 될 수 없는 클래스를 “추상 클래스”라고 한다. 추상 클래스는 일반화를 하기 위한 목적으로만 사용된다.
- 포유류 a = new 포유류(); 는 잘못된 문장이다. 포유류는 인스턴스가 될 수 없기 때문이다.

21. 자동차의 상속 관계

- 자동차는 이탤릭 체
- 이탤릭 체로 표현하면 추상 클래스
- 자동차는 부모 클래스
- 버스, 스포츠카, 포크 레인은 자식 클래스
- 버스, 스포츠카, 포크 레인을 일반화 한 것 은 자동차
- 자동차를 확장한 것은 버스, 스포츠카, 포크레인
- 상속 = 일반화 + 상속



22. ia a 혹은 kind of 관계

- 버스는 자동차다.
- 스포츠카는 자동차다.
- 포크 레인은 자동차다.

```
자동차 c1 = new 버스();
자동차 c2 = new 스포츠카();
자동차 c3 = new 포크레인();
```

눈 앞에 버스가 있다고 생각해 봅시다. 해당 버스를 가리키면서 이렇게 말합니다. 자동차 **c1**이야.라고 말입니다. 틀린말인가요?

23. 레퍼런스 타입이 가지고 있는 기능만 사용할 수 있다.

- 버스 c1 = new 버스();

  c1.달리다();

  c1.안내방송하다();

- 자동차 c2 = new 버스();

  c2.달리다();

  c2.안내방송하다(); (X)

24. 어떤 자동차를 가지고 가도 대신 주차를 해주는 식당이 있다면?

- 스포츠카, 버스, 포크레인 등 어떤 자동차를 가지고 가도 대신 주차를 해주는 식당이 있다고 가정 해보자.
- 스포츠카를 대신 주차해 달라고 부탁하니 지붕을 여닫는 주차 요원을 봤다면?
- 버스를 대신 주차해 달라고 부탁하니 버스 안의 안내 방송을 실행하는 주차 요원을 봤다면?
- 포크레인을 대신 주차해 달라고 했더니 주차하러 가는 도중에 땅을 파고 있는 주차 요원을 봤다면?

자동차의 주인은 자동차의 기본 기능만
이용하여 주차를 하길 원한다.

25. 다음 박스에 올 수 있는 것은?

자동차 **c1 =** **new 스포츠카();**
c1.달리다**();**

자동차 **c2 =** **new 버스();**
c2달리다**();**

자동차 **c3 =** **new 포크레인();**
c3달리다**();**

레퍼런스 타입만 봐서는 어떤 인스턴스가 사용될지 알기가 어렵다.

26. 조상 type의 레퍼런스 변수는 후손 인스턴스를 레퍼런스 할 수 있다.

- 아무것도 상속받지 않은 클래스는 자동으로 **Object**를 상속받는다.
- 결국 최상위 클래스는 **Object**이다.
- **Object a = _____;**

밑줄에 올 수 있는 내용은

new 자동차(); (X)

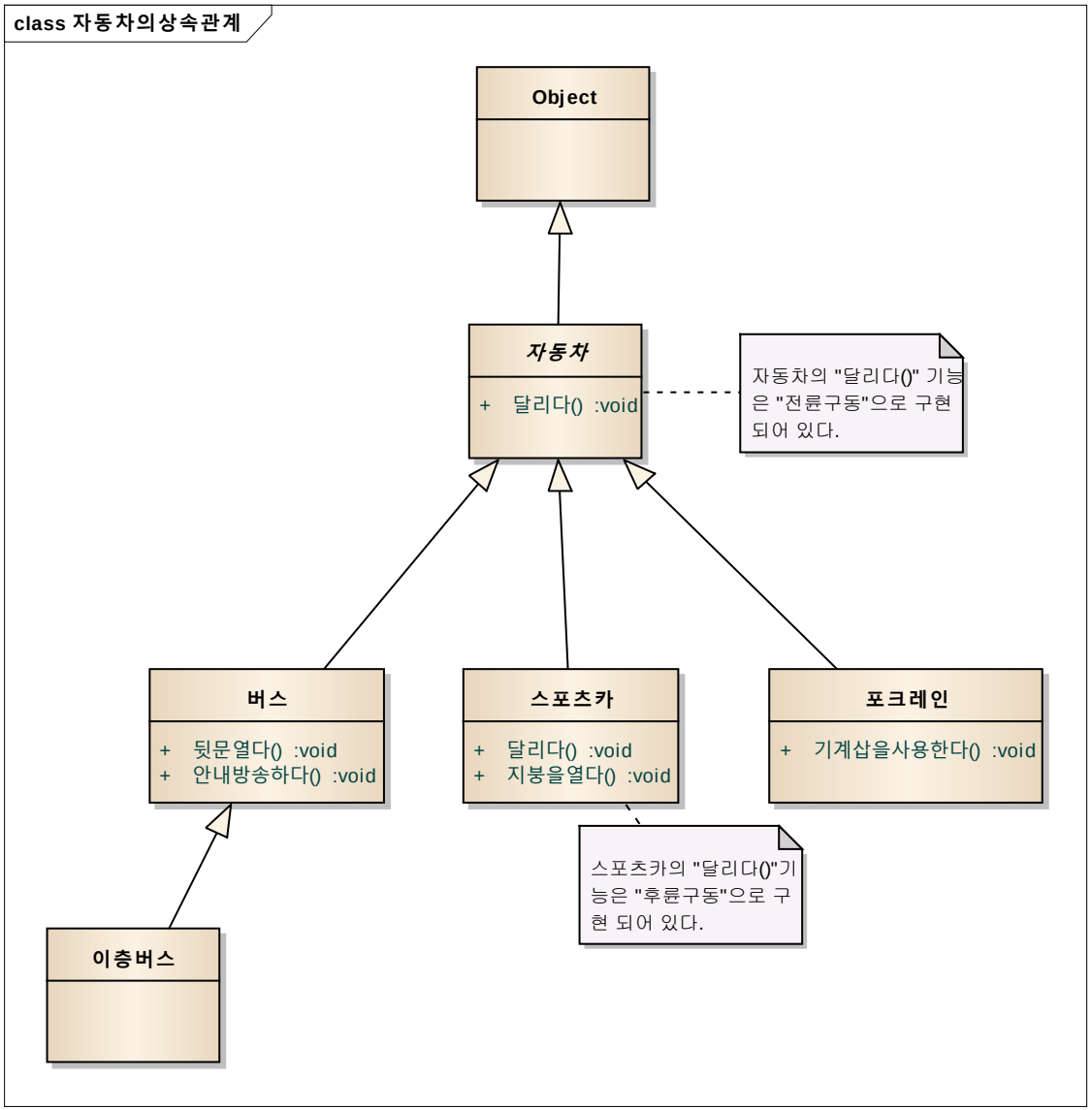
new 버스(); (O)

new 스포츠카(); (O)

new 포크레인(); (O)

new 이층버스();(O)

- **Object**가 가지고 있는 메소드만 사용가능하다.



27. Override. 올라타다. 겹치다. 중복되다.

- 버스 c1 = new 버스();

  c1.달리다();

위의 내용이 실행되면 버스의 부모클래스인 자동차가 가지고 있는 달리다() 메소드가 호출된다. 자동차의 달리다() 메소드는 전륜 구동으로 구현되어 있기 때문에 전륜 구동으로 동작하게 된다..

- 스포츠카 c3 = new 스포츠카();

  c3.달리다();

자동차, 버스, 포크레인, 스포츠카 모두 달리다() 라는 기능이 필요하다. 다만, 스포츠카의 경우 후륜 구동으로 다시 구현되었는데, 이렇게 부모와 완전히 같은 모양으로 다시 선언하는 것을 오버라이딩이라고 한다. 다시 선언되어 있기 때문에 이 경우 달리다는 후륜 구동으로 동작하게 된다.

28. 메소드가 오버라이딩 되면 무조건! 자식의 메소드가 실행된다.

- 스포츠카의 달리다 메소드는 오버라이딩 되었다고 하였다. 그렇다면 스포츠카는 전륜구동으로 달리겠는가? 후륜구동으로 달리겠는가?
- 여러분 눈 앞에 스포츠 카가 있다고 생각해 보자. 해당 스포츠카를 가리키면서 “자동차다.” 라고 말한다면 틀린말인가? 맞는 말인가?
- 위의 말에 이어서, 다음과 같이 말했다. “그 자동차는 달린다.” 그렇다면 그 자동차는 어떻게 달리고 있을까? 전륜구동일까? 후륜구동일까?
- 위의 말을 자바 코드로 바꾼다면 다음과 같다.

  ```
  자동차 c1 = new 스포츠카();

  c1.달리다();
  ```

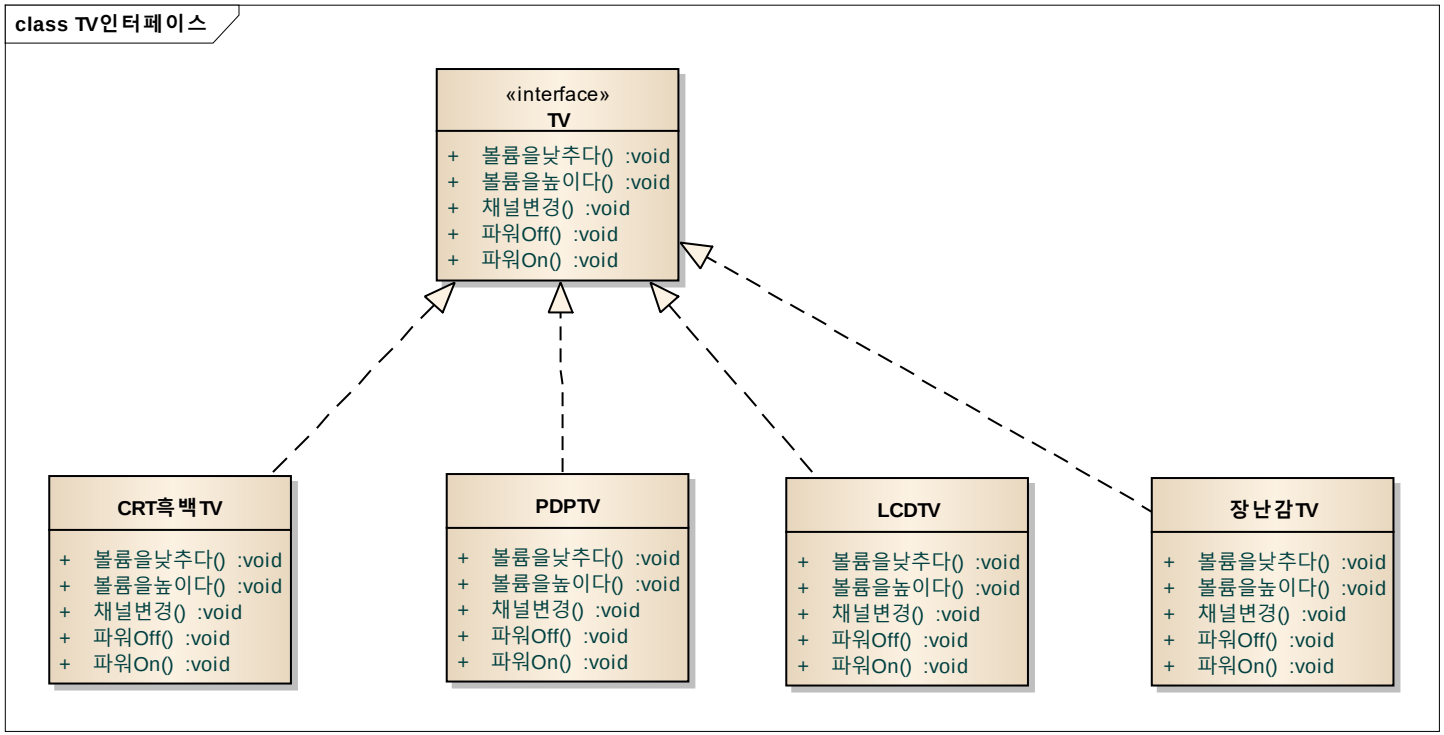
- 레퍼런스 타입이 자동차이기 때문에 자동차가 가지고 있는 메소드밖에 사용하지 못한다. 하지만, 여기에서 중요한 것은 실제 레퍼런스 변수가 레퍼런스 하는 인스턴스가 무엇이냐가 중요하다. 여기에서는 인스턴스가 스포츠카로 스포츠카는 달리다라는 메소드를 오버라이딩하고 있기 때문에 후륜구동으로 동작한다.

29. 소프트웨어 개발 라이프 싸이클과 인터페이스

- 소프트웨어를 만들기 위해서는 절차가 필요하다.
- 고객이 원하는 요구 사항을 잘 분석하여 구현을 해야 한다.
- 요구 사항에는 크게 기능적 요구 사항과 비 기능적 요구 사항이 있다.
- 예를 들어 블로그를 만든다면, 블로그 등록, 블로그 수정, 블로그 조회 등이 기능적 요구사항이라고 말할 수 있을 것이다.
- 이러한 기능을 잘 정리한 이후에 개발에 들어갈 수 있을 것이다.
- 어떤 제품이든지 간에 요구 사항을 잘 정의한 이후에 설계를 하고 그 설계한 것을 토대로 인스턴스를 만들게 되는 것이다.
- 앞에서 설계도는 클래스와 유사하다고 하였다. 그렇다면 요구사항은 어떻게 표현할까? 이런 기능적 요구사항은 인터페이스(interface)로 표현한다.

30. interface vs class

- TV를 만들기 전에 TV가 필요한 기능을 생각해 보자. 아마도, 전원에 대한 On, Off기능과 채널 변경 기능, 볼륨 조정 등이 필요할 것이다.



구현의 관계

31. 인터페이스가 가지고 있는 기능은 모두 선언만 된 것이다.

- 인터페이스가 가지고 있는 기능은 모두 선언만 된 것이다. 선언만 되어 있다는 것은 구현되어 있지 않다는 것을 의미한다.
- 추상 클래스는 일 부 메소드가 미구현일 수 있다.
- 클래스가 가지는 모든 메소드는 구현되어 있어야 한다. 구현되지 않은 메소드가 있을 경우 추상 클래스로 선언해야 한다.
- 인터페이스와 추상클래스는 인스턴스가 될 수 없다.

32. 인터페이스 타입도 레퍼런스 타입이 될 수 있다.

- TV t1 = new PDPTV();
- TV t2 = new LCDTV();
- TV t3 = new CRT흑백TV();
- TV t4 = new TV(); (X)
- TV t5 = new 장난감TV();

- 인터페이스와 클래스는 구현의 관계이다. 버스,스포츠카,포크레인은 모두 자동차의 한 종류라고 말할 수 있지만 LCDTV, PDPTV, CRT흑백TV, 장난감 TV는 TV 의 인터페이스를 구현하고 있다고 말할 수 있다.

자바 기본 프로그래밍

1. 자바 소개

목차

❑ 객체지향언어(Object Oriented Language)

- 60년 말 Simula 언어에서 발전

❑ C++를 문법을 기본으로 개발

- Java 이전에 객체지향언어로 가장 범용적인 언어
- 1983년 경, AT&T연구소의 Bjarne Stroustrup이 개발
- C언어에 객체지향 특성 확장

❑ 미국 Sun Microsystems사에서 개발한 객체지향 프로그래밍 언어

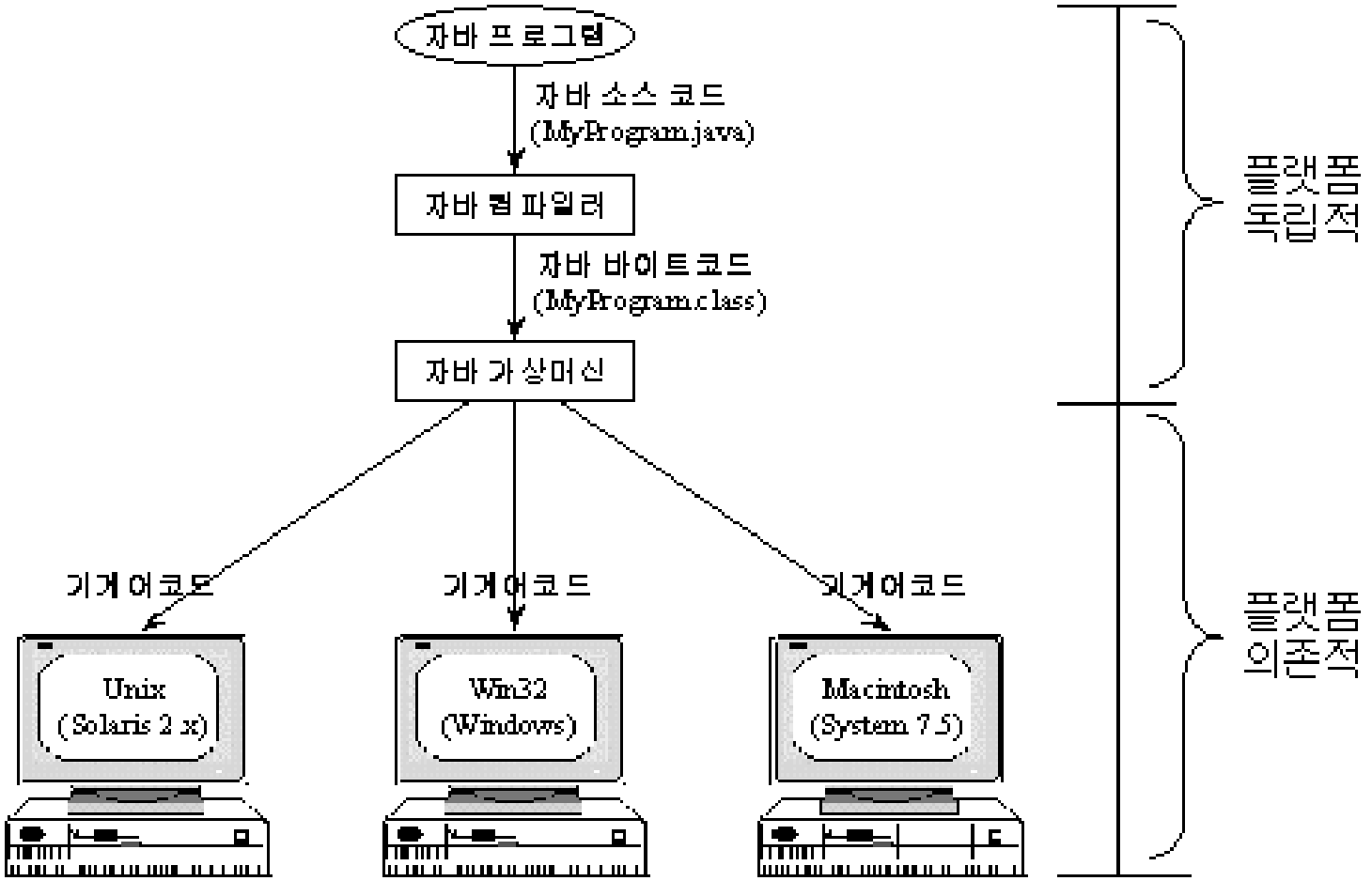
- 1995년 5월, Sun World에서 공식발표
- 1996년 1월, JDK1.0 발표
- 1998년, JDK 1.2 발표 1.0과 1.2의 차이가 너무 커서 이때부터 JAVA 2라고 불리기 시작함
- 2000년, JDK 1.3, HotSpot(Sun에서 만든 JIT구현), JNDI(디렉토리와 이름으로 서비스를 찾는 것) 포함
- 2002년, JDK 1.4, JCP에 의해서 오픈소스 정책으로 자바가 관리되기 시작함, Java 2 Security 모델의 확립(Sandbox), Java Web Start포함
- 2004년, JDK 1.5, 기능적으로 가장 많은 변화가 생긴 버전, Generics, annotation, auto boxing, enum, foreach, static import도입, java.util.concurrent API, scanner class
- 2006년, JDK 1.6, 기능에 별 차이없음. 보안,성능강화주력, JVM/Swing에서 퍼포먼스 향상, G1(Garbage First) GC도입

- 2011년, JDK 1.7, JVM : Dynamic Language support, Switch문에서 String사용가능, try-resource, 제네릭에서 타입추론, 숫자에서 underscore사용, concurency api강화, nio강화, sort강화, crypto 강화, gpu강화, JAVA FX가 기본으로 포함
- 2014년, JDK 1.8, 오라클로 인수된 후 첫번째 버전, JDK 1.5이후 가장 큰 언어적 변화(lambda및 함수형 프로그래밍, default method)가 생겼고 러닝커브가 크다. JEP에 의해서 새로운 기능들이 발의되기 시작, Nashorn(JS엔진), new Date and Time api, stream api, collection에 대한 함수형화(interface에 default가 생김으로써 가능), 병렬처리에 적합한 구조로 진화
- 2017년 Java 9, Java9이후 버전 번호는 출시년도와 달을 합친 숫자로 표현될 예정입니다. 예를 들어 2018년 3월에 출시한 Java 10은 Java 10 (18.3)이 됩니다.
- Java 9이후부터는 버전업 주기는 6개월 마다 이뤄집니다. 그 사이에는 버그 수정과 보안에 대응하는 마이너 버전이 제공될 예정입니다.

❑ 자바 언어는 단순하며 상대적으로 배우기 쉽다.

- 자바 언어는 C/C++ 언어의 구문과 C++ 언어의 객체지향성을 채택하면서 포인터, 연산자 중복 등의 복잡한 기능은 제외시킴
- 동적으로 할당되는 메모리를 프로그래머가 신경 쓸 필요 없이 Garbage Collector를 통해 자동으로 메모리를 관리해줌
- 예외처리(Exception Handling) 개념 도입
- 임의의 데이터베이스와 쉽게 연결하기 위한 JDBC(Java Database Connectivity) 제공

❑ 자바 언어는 플랫폼 독립적이다.



❑ 객체 지향 프로그래밍(Object-Oriented Programming: OOP)

– **개념 (Concept)**

- 서로 협동하는 다수의 에이전트 혹은 요원(객체; Object)들로 이루어진 프로그램
- 다수의 요원(객체; Object)들은 자신의 임무를 어떻게 처리할지(메소드; Method)와 자기 자신에 대한 정보(인스턴스 변수; Instance Variable)를 스스로 알고 있음.

– **핵심 용어 (Key Words)**

- 객체(Object): 사람, 장소, 사물 등 → 객체는 변수와 메소드를 가짐
- 클래스(Class): 같은 종류의 객체에 대한 범주 → 같은 타입(클래스)의 객체는 같은 변수와 메소드를 가짐.
- 메소드(Method): 객체에 의해 행해지는 행동
- 상속(Inheritance): 클래스 구현에서 일반화를 제공 → 간결한 코드, 객체의 재사용에 용이함
- 다형성(Polymorphism): 서로 다른 객체가 동일한 메소드에 대하여 다른 행동을 하는 것 (overloading)

– **장점:**

- 객체의 재사용으로 인한 개발 시간과 비용 감소
- 클래스에 대한 독립적인 테스트로 디버깅이 용이함

3. 자바 플랫폼이란?

❑ 자바 프로그램의 동작 방식

- 소스 코드 작성
- 중간 코드(Bytecode)로 컴파일 (Compile)
- 기계어로 인터프리팅 (Interpreting)
- 자바 가상 머신 (JVM)에서 실행

❑ 컴파일 (Compile)

- 컴파일러를 이용하여 고급언어를 기계어로 번역하는 과정
- 프로그램 실행 전에 미리 수행되어야 함
- 어셈블리 코드와 비슷하지만 플랫폼 독립적이고 Portable 함

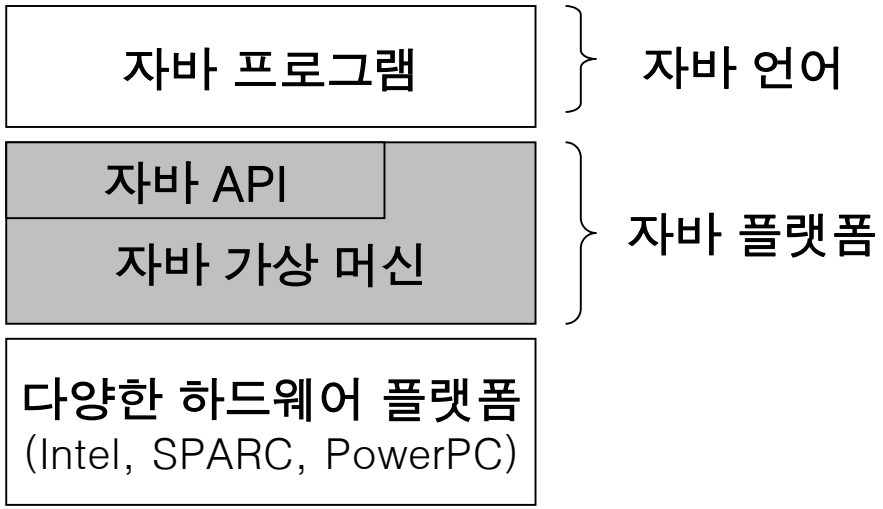
❑ 인터프리팅 (Interpreting)

- 중간코드를 각 하드웨어에 따른 기계어로 번역
- 실행시간(Run-time)에 번역됨
- 자바 가상 머신(Java Virtual Machine)에 의하여 수행됨

❑ 자바 언어: 컴파일러와 인터프리터를 모두 사용

❑ 자바 가상 머신 (JVM)

- 자바의 중간코드(Bytecode)를 실행시키는 소프트웨어로 만들어진 CPU
- 중간코드(Bytecode)는 컴퓨터 하드웨어에 무관하게 공통으로 사용할 수 있는 자바 전용 어셈블리 언어



자바 기본 프로그래밍
2. 자바 개발환경

목차

❑ JDK

- 자바 언어를 개발한 썬 마이크로시스템사는 자바 프로그램 개발을 위한 자바 개발환경(JDK: Java Development Kit) 를 제공
- 세가지 버전 제공 Java SE(Standard Edition), Java EE (Enterprise Edition), Java ME (Micro Edition)

❑ JDK 발전과정

- 1995년 최초 JDK 1.0 공개
- JDK1.2 부터 SDK 라 불렸고 J2SDK(Java 2 SDK) 라고 한다.
- JRE(자바실행환경) 은 SDK에 자바 컴파일러와 디버깅 도구등 몇 개의 도구가 제외된 버전으로 일반 사용자를 타켓으로 한다.
- Java SE Development Kit 10이 최신 버전. 하지만, 업계에서는 JDK 5, JDK8이 가장 많이 사용되고 있습니다.

❑ JDK 설치

- Java SE Development Kit 8다운로드
- 설치파일 실행
- 환경변수 설정(JAVA_HOME, CLASSPATH, PATH)
- 설치 확인
  - javac -version
  - java -version

❑ JDK Document 설치와 사용법

- 자바 개발 환경 설치로 자바가상 머신과 API가 설치 되었다.
- API은 썬에서 제공하는 자바 기본 클래스 라이브러리로 온라인상에서 http://docs.oracle.com/javase/8/docs/api/ 통해 참고 할 수 있다.
- 온라인이 아닌 언제든지 자신의 PC에 설치하고 참고할 수 있다.

  JDK 다운로드 페이지에 Aditional Resource에서 다운로드 후 압축을 풀면 된다.

자바 기본 프로그래밍

3. 자바 프로그램 작성

목차

- 보통 자바 프로그램은 하나의 클래스 또는 여러 개의 클래스로 이루어진다.
- 클래스 내부에는 실행에 필요한 변수나 메서드 등을 정의한다.

```
public class 클래스 이름 {
// 변수 정의
// 메서드 정의
}
Class 클래스 이름 {
// 변수 정의
// 메서드 정의
}
```

❑ 자바 주석문

주석문은 프로그램을 설명하기 위해 사용하는 문장으로 프로그램을 수행하는 데 아무 영향을 미치지 못한다.

```
/**
 *  파일명: Comment.java
 */
public class Comment {

   /* 첫 번째 출력될 내용을 저장한 변수 선언 */
   static String s = "Hello Java";

  // 두 번째 줄에 내용을 출력하는 메서드 선언
  static string getMessage() {
     return "Hello Java";
  }
  public static void main( String args[] ){
      System.out.println(s + getMessage()); // 콘솔 출력
  }
}
```

2. 자바 어플리케이션 실행

- 자바 어플리케이션은 바이트 코드로 번역된 후에 바로 실행할 수 있는 프로그램
- **java.exe** 명령만 있으면 언제든지 실행 할 수 있다.
- 자바 어플리케이션은 클래스 내에 반드시 **main** 메서드를 정의 해야 한다.

(숙달되기 전 까지 꼭 외우세요)

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

실습1. 첫번째 자바 프로그래밍

❑ HelloWorld.java

– 소스코드 작성

1. Editor를 사용하되 Copy 하지 말고, 오른쪽 상단의 소스코드를 작성하여 "HelloWorld.java"로 저장하라.

– 컴파일

1. 명령 프롬프트를 실행하여 소스코드가 저장된 디렉토리로 이동하라
2. `javac HelloWorld.java`
3. HelloJAVA.class 생성되었음을 확인하라. (`dir`)
4. HelloJAVA.java를 오른쪽 하단과 같이 수정해서 그대로 저정한 후, 1~3을 하라.

– 실행

1. `java HelloWorld`
2. `java HelloWorld.class`

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

실습1. 첫번째 자바 프로그래밍 - 해설

```
public class HelloWorld{

   public static void main(String[] args){

         System.out.println("Hello World!");

   }

}
```

- 일반적으로 하나의 자바 소스코드 파일에는 하나의 자바 클래스가 들어있음
- 클래스 이름은 대문자로 시작하는 것이 관례
- 일반적으로 소스코드 파일의 이름과 클래스 이름이 동일하게 함
- 자바 인터프리터는 자바 프로그램의 시작 지점으로 **main** 함수를 찾음
- **main** 함수 내의 각 명령을 순차적으로 실행
- **System.out.println** 메소드는 표준 출력으로 명시된 문자열을 화면에 출력하는 역할을 함

자바 기본 프로그래밍

4. 자바 기본 구문

목차

1 식별자와 예약어

❑ 식별자(Identifier)

- 프로그래머가 직접 만들어줘야 하는 이름
- 예: 변수명, 클래스명, 메소드명 등
- 대문자, 소문자, 숫자, _, $등, 숫자는 식별자의 첫 문자로 사용될 수 없음

❑ 예약어(Keyword)

- 프로그래밍 언어에 미리 정의된 의미있는 단어
- 예약어는 식별자로 사용하지 않음

abstract	double	int	strictfp
boolean	else	interface	super
break	extends	long	switch
byte	final	native	synchronized
case	finally	new	this
catch	float	package	throw
char	for	private	throws
class	goto *	protected	transient
const *	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while

2. 변수(Variable)와 상수(Constant)

❑ 값(Value)

- 데이터(Data) 자체
- 예: 1, 0, 20.5, true, 'a'

❑ 자료형(Type)

- 데이터(Data)의 종류
- 예: 정수(Integer), 실수(Float, Double), 문자(Character), 논리(Boolean – True/False), 문자열(String) 등
- Java의 모든 데이터 값들은 타입을 가짐
  - 저장될 메모리와 저장 공간 결정
  - 해당 값에 적용될 연산을 결정
  - 프로그램의 에러 점검

2. 변수(Variable)와 상수(Constant)

❑ 변수(Variable)

- 데이터를 저장할 메모리의 위치를 나타내는 이름 → 데이터 자체는 아님
- 변수의 선언 (Declaration)
  - 변수를 사용하기 위해서는 사용 전에 미리 선언해야 함
  - **Type Variable1, Variable2, …;**
    - `int numberOfItems, numberOfTitles;`
    - `boolean isHoliday;`
- 초기화(Initialization)
  - 변수에 최초의 값을 할당하는 것

```
1. numberOfItems = 0; // 오른쪽 표현의 값을 왼쪽 변수에 할당
2. isHoliday = false;
```

  - 보통 변수의 선언과 동시에 초기화를 함

```
1. float averageOfScores = 65.4;
2. String companyName = "(주)써니베일";
```

  - 초기화 후에 값이 변할 수도 있음 → 變數
- 같은 타입의 데이터만 저장할 수 있음
  - `isHoliday = 1; // COMPILE TIME ERROR`

2. 변수(Variable)와 상수(Constant)

❑ 상수(Constant)

- 변경될 수 없는 고정된 데이터 → 常數
- 코드의 이해와 변경이 쉬움
- 분산된 상수로 인한 에러를 방지
- 정의
  - "final" 키워드를 사용해서 대문자로 표현함
    - `final double PI = 3.1416;`
    - `final int MAXIMUM_SPEED = 240;`

3. 자바의 자료형

❑ 기본 자료형(primitive data types)

- 최소 단위의 자료형
- 다른 자료형으로 분해되지 않음
- 메소드가 없이 값만 가짐
- int, float, double, char 등

❑ 참조 자료형(reference data types)

- 여러 자료형들의 집합으로 구성된 클래스의 객체를 참조
- 데이터와 메소드를 가짐
- String, Vector, Integer 등

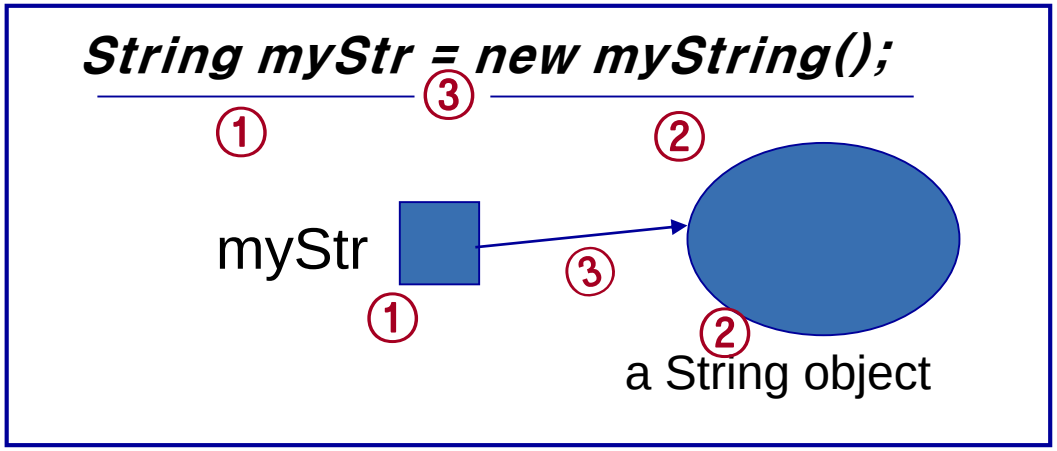
3. 자바의 자료형 – 기본 자료형

자료형	키워드	크기	표현 범위	사용 예
논리형	boolean	1bit	true OR false(0과 1이 아니다)	boolean isFun = true;
문자형	char	2byte	0~65,535	char c = 'f';
정수형	byte	1byte	-128 ~ 127	byte b = 89;
	short	2byte	-32,768 ~ 32,767	short s = 32760;
	int	4byte	-2147483648:2147483647	int x = 59; int z = x;
	long	8byte	…	long big = 3456789;
실수형	float	4byte	-3.4E38 ~ 3.4E38	float f = 32.5f
	double	8byte	-1.7E308 ~ 1.7E308	double d = 23.34

3. 자바의 자료형 – 기본 자료형 vs. 참조 자료형

int myInt = 19;

myInt **19**



	기본 자료형	참조 자료형
변수값	실제값	Object 참조값
정의방식	Java 내부에 이미 정의됨	클래스 정의
생성방식	"19", "3.14", "true"	"new"*
초기화방식	default	생성자(Constructor)*
사용방식	연산자 ("+ ", "–", "*", "/" 등)	메소드

** object 생성에 대해서는 5장에서 다룸*

4. 자료형의 변환

❑ 자료형 변환

- 하나의 자료형을 다른 자료형으로 바꾸는 행위
- 자료형의 변화 없이 하나의 변수에서 다른 변수로 값 할당 불가
  - **boolean isHoliday = 1; // COMPILE TIME ERROR**

❑ 변환 규칙

- 암묵적 자료 유형 변환 (implicit casting): 자료의 범위가 좁은 자료형에서 넓은 자료형으로의 변환은 시스템이 자동으로 행함
- 명시적 자료 유형 변환(explicit casting): 자료의 범위가 넓은 자료형에서 좁은 자료형으로의 변환은 시스템에서 자동으로 수행할 수 없으며 프로그래머가 강제로 변환해야 함

```
int i = 101;
long l = 101L;
l = i; // No Problem
i = l; // Compiler ERROR.
i = (int) l; // Now, it's OK.
```

참고 – 강형언어와 약형언어

❑ 강형 언어 (Strongly-typed language)

- 모든 변수의 형 검사가 컴파일 시에 이루어지는 언어
- 신뢰성, 유지보수성, 가독성이 높음
- C, Java, Ada 등

❑ 약형 언어 (Loosely-typed language)

- 정적인 형 체계를 갖추지 않은 언어
- 실행시에 변수가 가질 자료의 유형이 결정됨
- LIST, APL, SNOBOL 등

실습1 – Data Type

❑ 괄호 안에 적절한 데이터 타입을 기술하라.

```
public class Student {
   (     ) stNumber;       // 학번
   (     ) stName;         // 이름
   (     ) isEnrolled;     // 등록 여부
   (     ) grade;          // 평점
   (     ) address;        // 주소
   (     ) major;          // 전공
   (     ) unit;           // 이수 학점
   (     ) haveMinor;      // 부전공 여부
   (     ) juminNo;        // 주민번호(-없이 13자리숫자)
   (     ) cellphone;      // 핸드폰 번호(-포함한 숫자)
   (     ) age;            // 나이
   (     ) email;          // 이메일주소
}
```

실습2 – Typecast1.java

❑ Typecast.java 를 컴파일하고 실행 결과를 확인하라. 이 때 에러가 있으면 수정하라.

```
public class Typecast1{
   public static void main(String[] args)
   {
          byte b = 25;
          short s = b;
          int i = s;
          long l = i;
          float f = i;
          double d = f;

          byte b_1 = 256;

          System.out.println("b = " + b);
          System.out.println("s = " + s);
          System.out.println("i = " + i);
          System.out.println("l = " + l);

          System.out.println("f = " + f);
          System.out.println("d = " + d);
          System.out.println("b_1 = " + b_1);
   }
}
```

실습3 – Typecast2.java

- Typecast2.java 를 컴파일하고 실행 결과를 확인하라. 이 때 에러가 있으면 수정하라.

```
class Typecast2 {
        public static void main(String args[]){
                byte b;
                int i = 414;
                float f = 123.456;

                b=(byte)i;
                System.out.println("int 414를 byte로 변환 : " + b);
                i=(int)f;
                System.out.println("float 123.456을 int로 변환: " + i);
                b=(byte)f;
                System.out.println(" float 123.456을 byte로 변환: " + b);
        }
}
```

5. 연산자 – 산술연산자와 증감연산자

❑ 산술 연산자

- 사칙 연산 (+, -, *, /)
- 나머지 연산 (%)
- - 연산

❑ 증감 연산자

- ++a, --a
- a--, a++

```
public class Arith
{
  public static void main(String[] args)
  {
            int a = 7;
            int b = 3;

            System.out.println(a+b);
            System.out.println(a-b);
            System.out.println(a*b);
            System.out.println(a/b);
            System.out.println(a%b);
            System.out.println(-a);
            System.out.println(++a);
            System.out.println(--b);
            System.out.println(a++);
            System.out.println(b--);
  }
}
```

5. 연산자 – 산술연산자와 증감연산자

❑ 숫자의 나누기 연산 (/)

– 내부적으로 '버림'에 의한 형변환이 이루어짐

- `10.0 / 3.0 → 3.3333333333333335`
- `10.0 / 3 → 3.3333333333333335`
- `10 / 3.0 → 3.3333333333333335`
- `10 / 3 → 3`
- `(double)10 / 3 → 3.3333333333333335  // 10은 내부적으로 double로 자동 형변환됨`
- `10 / (double) 3 → 3.3333333333333335  // 3은 내부적으로 double로 자동 형변환됨`
- `(double)(10/3) → 3.0  // (10/3) 전체가 내부적으로 double로 자동 형변환됨`

❑ 숫자의 나머지 연산 (%)

– 주로 정수(int) 값을 대상으로 하여 결과도 정수임

– 나누어지는 수에서 가장 가깝게 몫을 구한 후 나머지를 계산

- 나머지 값이 양수, 음수 모두 가능
- 수학의 나머지 정의(n÷d=q…r → r은 0<r<d 인 자연수)와는 다름
- `44%3 → 2`
- `-7 % 3 → -1`
- `7 % -3 → 1`

5. 연산자 - 관계 연산자와 논리 연산자

❑ 관계 연산자

- 같음(==), 다름(!=)
- 보다큼(<), 보다작음(>), 크거나같음(<=), 작거나같음(>=)
- 결과는 Boolean 타입(true 또는 false)

❑ 논리 연산자

- AND(&&), OR(||), NOT(!)
- 결과는 Boolean 타입

a	b	a&& b	a || b	!a
False	False	False	False	True
False	True	False	True	
True	False	False	True	False
True	True	True	True	

5. 연산자 - 관계 연산자와 논리 연산자 [연습문제]

```
int a = 7;

int b = 3;

a==b;  // True or False?

a!=b; // True or False?

a<b; // True or False?

a>b; // True or False?

(a>b)&&(a<b); // True or False?

(a>b)||(a<b); // True or False?

boolean x = 7 == 9; // x값은 True or False?

x = 9.5 >= 9.5; // x값은 True or False?

x = 8 != 9 - 1; // x값은 True or False?

x = false == (10 == 0); // x값은 True or False?
```

5. 연산자 - 비트 연산자

❑ 비트 논리 연산자

- AND(&), OR(|), XOR(^), NOT(~)

❑ 비트 이동 연산자

- 우측으로 n비트 이동(>>n)
- 좌측으로 n비트 이동(<<n)

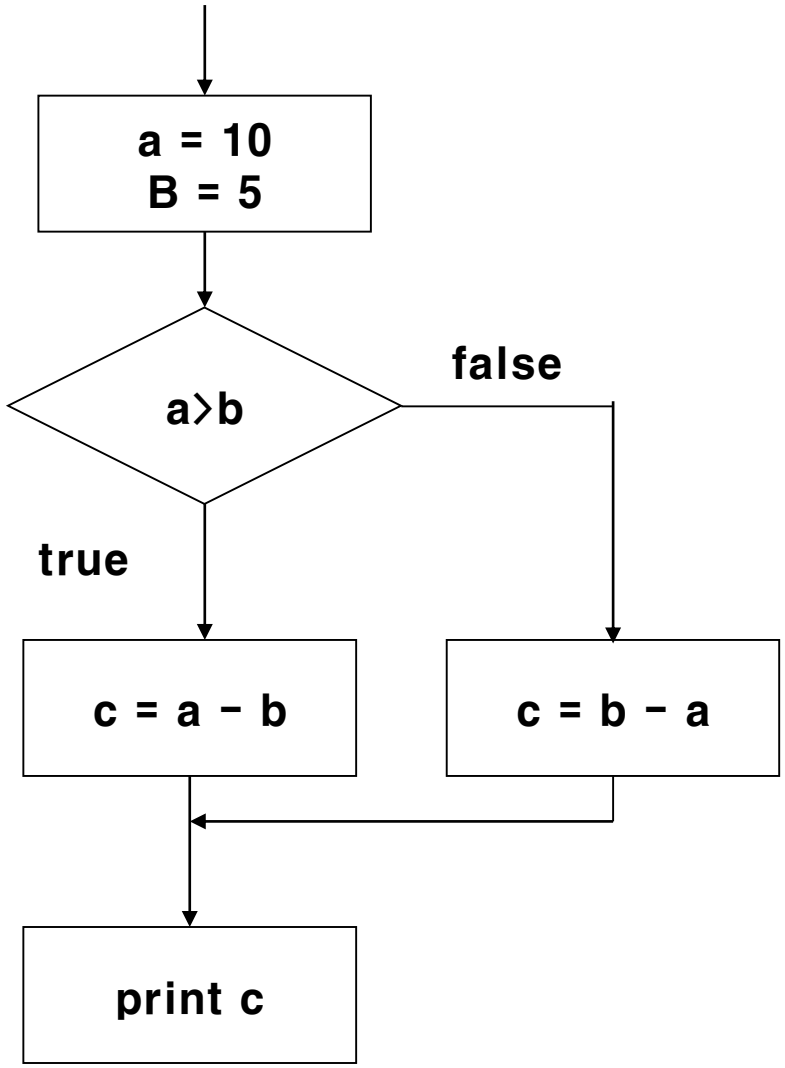
A	**0000 1111**
B	**1111 0000**
A & B	**0000 0000**
A | B	**1111 1111**
A ^ B	**1111 1111**
~A	**1111 0000**
A>>3	**0000 0001**
A<<3	**0111 1000**

참고 - 연산자 우선순위

순위	연산자	분류이름	동일시 계산 우선
1	. [ ] (params) expr++ expr--	postfix increment, decrement	왼쪽에서 오른쪽으로
2	++expr --expr +expr -expr ! ~	prefix increment, decrement plus sign, minus sign not, bit complement	오른쪽에서 왼쪽으로
3	new (type)expr	Creation or cast	오른쪽에서 왼쪽으로
4	* / %	Multiplicative	왼쪽에서 오른쪽으로
5	+ -	Addictive	왼쪽에서 오른쪽으로
6	<< >> >>>	Shift	왼쪽에서 오른쪽으로
7	< <= > >= instanceof	Relational	왼쪽에서 오른쪽으로
8	== !=	Equality	왼쪽에서 오른쪽으로
9	&	Bitwise AND	왼쪽에서 오른쪽으로
10	^	Bitwise XOR	왼쪽에서 오른쪽으로
11	|	Bitwise OR	왼쪽에서 오른쪽으로
12	&&	Logical AND	왼쪽에서 오른쪽으로
13	||	Logical OR	왼쪽에서 오른쪽으로
14	?:	Conditional	오른쪽에서 왼쪽으로
15	= += -= *= /= %= &= |= ^= >>= <<= >>>=	Assignment	오른쪽에서 왼쪽으로

❑ 기본 구문

```
if (condition) statement;

if (condition) {
      statement1;
      statement2;
}

if (condition) {
      statement1;
      statement2;
} else {
      statement3;
}
```



❑ 유형 I: 단순 if

```
if (year % 4 == 0) {
  System.out.println (year + "년은 윤년입니다.");
}
```

❑ 유형 II: If-else

```
if (year % 4 == 0) {
  System.out.println (year + "년은 윤년입니다.");
} else {
  System.out.println (year + "년은 윤년이 아닙니다.");
}
```

❑ 유형 III: 중첩 if

– 거짓인 경우에도 다른 경우를 잇달아 테스트함

```
if (year % 4 != 0) {
    System.out.println (year + "년은 윤년이 아닙니다.");
} else if (year % 100 != 0) {
    System.out.println (year + "년은 윤년입니다.");
} else if (year % 400 != 0) {
    System.out.println (year + "년은 윤년이 아닙니다.");
} else {
    System.out.println (year + "년은 윤년입니다.");
}
```

실습1. Swap.java

```
public class Swap
{
  public static void main(String[] args)
  {
        int n1 = 10;
        int n2 = 30;

        // Q1: n1값이 항상 큰 값이 되도록, 필요할 경우 n1과 n2를 교체하라.
        …
         System.out.println("n1: " + n1);
         System.out.println("n2: " + n2);


        // Q2:  아래에서 잘못된 것을 찾아서 수정하라.
        int a1 = 20;
        int a2 = 10;
        int a3 = 50;

        if (a1 >= a2) {
              if (a2 > a3) {
                    System.out.println("a1이 최대값입니다.");
              }
        } else {
              System.out.println("a1(" + a1 + ")은 a2(" + a2 + ")보다 작습니다.");
        }
  }
}
```

6. 조건문 – switch-case

❑ 기본구문

```
switch(expression){
   case value1:
        statement1;
        break;
   case value2:
        statement2;
        break;
   default:
        statement3;
        break;
}
```

- expression의 값에 해당되는 case를 수행
- 해당되는 값이 없을 경우 default 절을 수행
- switch-case 문에서의 break는 **switch** 문의 끝으로 프로그램의 흐름을 이동시킴

▪ switch

```
switch (month) {
   case 2:
          days = 28;
          break;
   case 4:
   case 6:
   case 9:
   case 11:
          days = 30;
          break;
   default:
          days = 31;
          break;
 }
```

▪ if

```
if (month == 2) {
    days = 28;
} else if ((month == 4) || (month == 6) ||
(month == 9) || (month == 11)) {
    days = 30;
} else {
    days = 31;
 }
```

6. 조건문 – switch-case

❑ switch-case 문에서의 break을 사용하지 않을 경우 다음 case 문이 계속해서 수행 됨

```
switch (month) {
   case 12:
         system.out.println("한 해를 마무리하는 12월입니다.");
   case 11: case 1: case2:
         system.out.println("여전히 겨울입니다.");
}
```

- 그러나 바람직하지 않은 프로그램 스타일

- switch 조건문에서는 앞 장처럼 default를 포함해서 가능한 모든 경우에 대한 case 절과 break문을 정의할 것

– month = 12:

"한 해를 마무리하는 12월 입니다."

"여전히 겨울입니다."

– month = 11:

"여전히 겨울입니다."

– month = 3:

아무 동작도 하지 않고 switch를 빠져나옴 (why?)

7. 반복문 - while

❑ 기본구문(Syntax):

```
while (condition) {statements}
```

- condition 값은 if문에서와 같이 boolean 값을 가짐
- condition 값이 참인 동안 해당되는 구문을 반복해서 수행함
- 초기 condition 값이 false인 경우 한번도 수행하지 않음
- condition 값이 false가 되면 while 문을 빠져 나옴
  - condition 값이 계속 true이면 무한히 반복함
  - 의도적으로 무한 Loop를 실행시키는 경우도 있음

```
// 1에서 10까지 숫자를 출력함
int x = 1;
while (x <= 10){
   System.out.println(x);
   x = x + 1;
}
```

실습1. PowerOfTwo.java

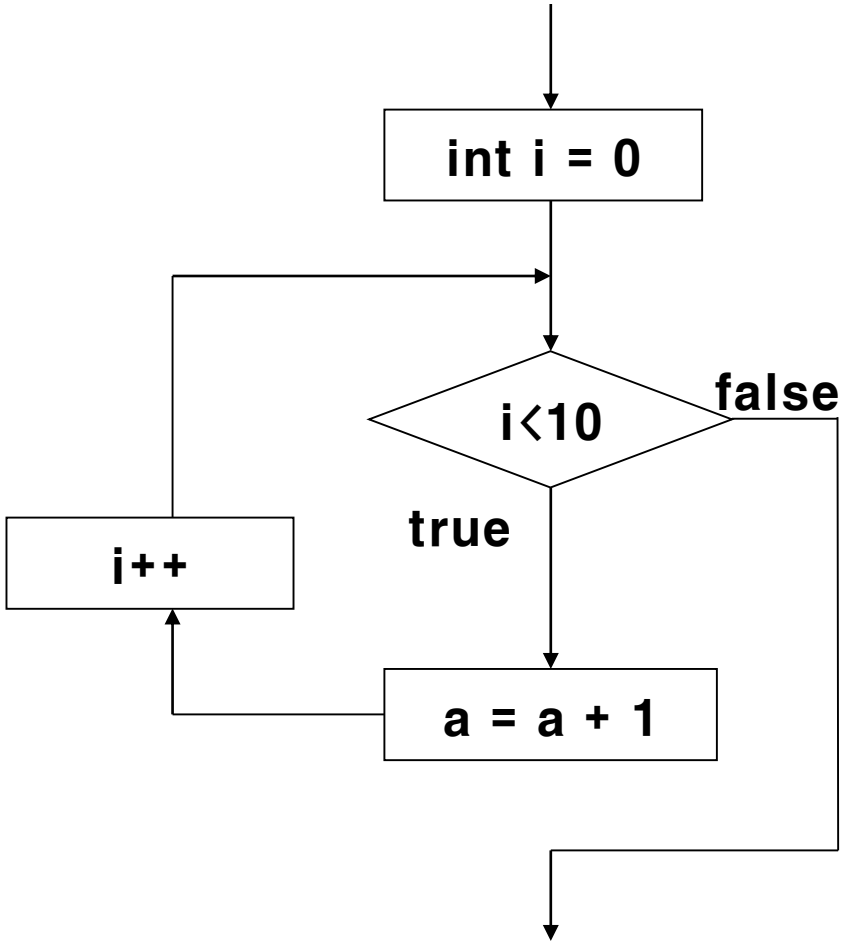
```
import java.util.*;  // Scanner 클래스를 사용하기 위함
                     // Scanner 클래스와 System.in, nextInt() 메소드는 추후 다룰 것임

public class PowerOfTwo
{
  public static void main(String[] args)
  {
        Scanner stdin = new Scanner(System.in);

        int result = 1;
        System.out.print("승수: ");
        int pow = stdin.nextInt();
        int n = pow;

        // Q: 아래 while 반복문을 작성하시오.
        while

                . . ●

        System.out.println("2의 " + pow + "제곱근은 " + result + "입니다.");
  }

}
```

실습2. PrimeDetect.java

```
/*****************************************************************************
 * Q: 키보드에서 입력을 받은 숫자가 소수(Prime number)인지 아닌지를 판단하려고 합니
 * 다. 2부터 시작해서 자신보다 작은 숫자까지 나누어 나머지 값이 0이 나오지 않을 때
 * 소수라고 결론을 내리려고 합니다. 입력받는 숫자는 양의 정수라고 가정하고 빈 부분을 작성하세요.
*****************************************************************************/
import java.util.*;  // Scanner 클래스를 사용하기 위함

public class PrimeDetect
{
  public static void main(String[] args)
  {
        Scanner stdin = new Scanner(System.in);
        System.out.print("소수인지 판단할 숫자: ");

        int divisor = 2;
        int num = stdin.nextInt();
        boolean isPrime = true;

        while
                . . .

        if ( . . . ) {
                System.out.println(num + "은 소수입니다.");
        } else {
                System.out.println(num + "은 소수가 아닙니다.");

        }
  }
}
```

❑ 세 가지 제어 요소

```
for( 초기화; 조건검사; 증감연산 ){
      statement1;
      statement2;
      ……
}
for( int i=0; i<10; i++ ){
      a = a + 1;
}
```



❑ for 반복문

– while 반복문을 간략화하여 사용 가능

```
Initialize;
while ( test )  {
   Statements;
   next;
}
```

```
for ( initialize;  test;  next )  {
   Statements;
}
```

```
int x = 1;
while ( x <= 10 ) {
  System.out.println( x );
  x = x + 1;
}
```

```
for (int x = 1; x <= 10; x = x + 1)
  System.out.println(x);
```

7. 반복문 – do-while

❑ do-whle 반복문

- while 반복문과의 차이점 → do 블록을 우선 한번 수행하고 반복 여부를 확인함
- 적어도 한번은 수행되어야 하는 반복 구문에 사용
- while 문에서 확인할 조건이 반복 구문 내에서 할당되는 경우

```
char ch;
do {
   ch = getCharFromFile;
   process input;
} while (ch != end-of-file);
```

7. 반복문 – break/continue

❑ switch-case 문에서의 break

- switch 문의 끝으로 프로그램의 흐름을 이동시킴
- break 문을 사용하지 않을 경우 다음 case 문이 계속해서 수행 됨

```
switch (month) {
    case 2:
      days = 28;
      break;
    case 4:
    case 6:
    case 9:
    case 11:
      days = 30;
      break;
    default:
      days = 31;
      break;
 }
```

```
switch (month) {
    case 12:
      system.out.println("한 해를 마무리하는 12월입니다.");
    case 11: case 1: case2:
      system.out.println("여전히 겨울입니다.");
 }
```

- 바람직하지 않은 프로그램 스타일
- switch 조건문에서는 앞 장처럼 default를 포함해서 가능한 모든 경우에 대한 case 절과 break문을 정의할 것

7. 반복문 – break/continue

❑ continue

- 반복문 내에서 continue를 만나면 continue 이후의 문장은 실행하지 않음
- 반복문의 검사 위치로 돌아가 반복문을 계속 수행
- 반복문에만 적용됨 → Switch 문에는 적용되지 않음
- 반복 블록을 벗어나지 않고 블록의 가장 마지막으로 이동하여 다시 반복 조건을 검사하도록 함

```
while ( month < 12 ) {
    statement1;
    switch (month) {
      case 2:
            days = 28;
            break;
      case 4: case 6: case 9: case 11:
            days = 30;
            continue;
      default:
            days = 31;
            break;
    }
    statement2;
    if (month == 0)
        break;
}
```

Q1) month가 2인 경우 실행되는 것은?

Q2) month가 4인 경우 실행되는 것은?

Q3) month가 0인 경우 실행되는 것은?

7. 반복문 – break/continue

❑ break

- 반복문을 종료하고 반복문을 벗어나 다음 문장을 실행
- 중첩된 반복문에서 한 단계씩 반복문을 벗어남
- 몇몇 반복문은 하나 이상의 반복 종료 시점을 가질 수 있음

```
char ch;
do {
    ch = getCharFromFile;
    process( s );
} while ( ch != end-of-file );
```

1. while 반복문

```
ch = getCharFromFile;
while ( ch != end-of-file )  {
    process input;
    ch = getCharFromFile;
}
```

2. while 반복문 & break

```
while (true) { // 무한 반복
   ch = getCharFromFile;
    if ( ch == end-of-file ) {
        break;
    }
    process input;  …
}
```

3. for 반복문 (최대 10회 반복 가정)

```
for (int i = 0; i < 10; i++) {
   ch = getCharFromFile
   if (ch == end-of-file) {
       break;
   }
   process(s);
}
```

7. 반복문 [연습문제]

❑ 다음 소스에서 잘못된 것은 무엇인가?

```
int[] intArray;
intArray = new int[5];
intArray[0] = 3;
intArray[1]= 6;
intArray[2] = 9;

int result = 0;
for (int i =0; i <= intArray.length; i++){
    result = result + intArray[i];
}
System.out.println("intArray의 4번째 값: " + intArray[3]);
```

❑ "for" vs. "while"

```
int i = 0;
while ( i < 10 ) {
    if ( condition( i ) ) {
        continue;
    }
    process( s );
    i++;
}
```

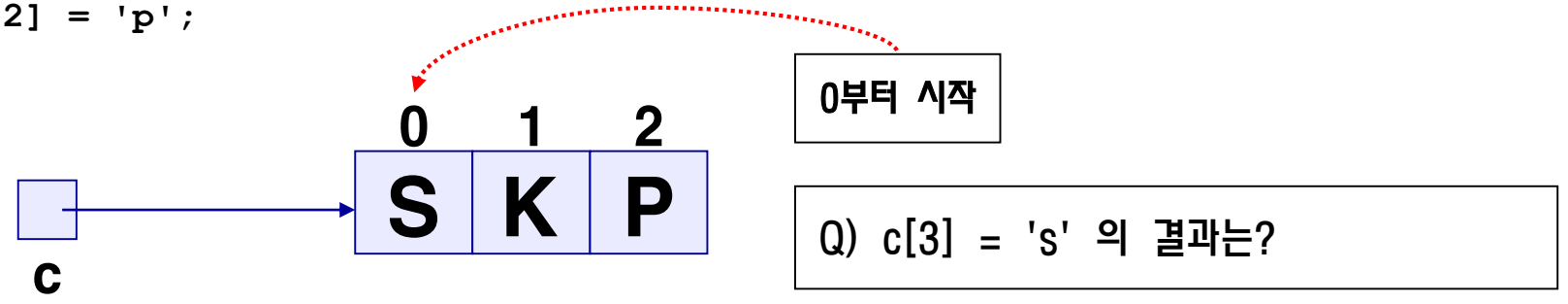
```
for ( int i = 0; i < 10; i++ ) {
    if ( condition( i ) ) {
        continue;
    }
    process(s);
}
```

Q) 위의 두 구문이 다른 점은 무엇인가?

❑ 배열

- 동일한 자료 유형의 여러 값들로 이루어진 객체(Object)
- 배열은 자체는 "new"로 생성되는 참조 자료형임
- 배열에 포함된 각 값들은 기본 자료형(Primitive Type)일 수도 있고, 다른 객체를 참조하고 있는 클래스형(Class Type)일 수도 있음

```
char[] c;              // 배열 객체를 참조하는 변수의 이름: c
c = new char[3];       // "new"는 char 유형의 데이터 4개를 담을 수 있는 메모리 공간을 할당함
                       // c 변수는 위에서 할당된 메모리 주소를 참조함
c[0] = 's';            // 문자(character) 's' 를 index 0에 저장함
c[1] = 'k';
c[2] = 'p';
```



실습1. Triangle.java

다음과 같이 출력되는 프로그램을 작성하라.

```
X
XX
XXX
XXXX
XXXXX
XXXXXX
```

실습2. Gugudan.java

다음과 같이 구구단이 출력되는 프로그램을 작성하라.

```
1 * 1 = 1       2 * 1 = 2       3 * 1 = 3       4 * 1 = 4       5 * 1 = 5       6 * 1 = 6       7 * 1 = 7       8 * 1 = 8       9 * 1 = 9
1 * 2 = 2       2 * 2 = 4       3 * 2 = 6       4 * 2 = 8       5 * 2 = 10      6 * 2 = 12      7 * 2 = 14      8 * 2 = 16      9 * 2 = 18
1 * 3 = 3       2 * 3 = 6       3 * 3 = 9       4 * 3 = 12      5 * 3 = 15      6 * 3 = 18      7 * 3 = 21      8 * 3 = 24      9 * 3 = 27
1 * 4 = 4       2 * 4 = 8       3 * 4 = 12      4 * 4 = 16      5 * 4 = 20      6 * 4 = 24      7 * 4 = 28      8 * 4 = 32      9 * 4 = 36
1 * 5 = 5       2 * 5 = 10      3 * 5 = 15      4 * 5 = 20      5 * 5 = 25      6 * 5 = 30      7 * 5 = 35      8 * 5 = 40      9 * 5 = 45
1 * 6 = 6       2 * 6 = 12      3 * 6 = 18      4 * 6 = 24      5 * 6 = 30      6 * 6 = 36      7 * 6 = 42      8 * 6 = 48      9 * 6 = 54
1 * 7 = 7       2 * 7 = 14      3 * 7 = 21      4 * 7 = 28      5 * 7 = 35      6 * 7 = 42      7 * 7 = 49      8 * 7 = 56      9 * 7 = 63
1 * 8 = 8       2 * 8 = 16      3 * 8 = 24      4 * 8 = 32      5 * 8 = 40      6 * 8 = 48      7 * 8 = 56      8 * 8 = 64      9 * 8 = 72
1 * 9 = 9       2 * 9 = 18      3 * 9 = 27      4 * 9 = 36      5 * 9 = 45      6 * 9 = 54      7 * 9 = 63      8 * 9 = 72      9 * 9 = 81
```

8. 배열 – 선언과 생성

❑ 선언

- intArraytype[] arrayName;
- intArraytype arrayName[];

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

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

❑ 생성

int[10] intArray; → 선언시 배열의 크기를 지정할 수 없음

int[] intArray; → 선언만 된 상태의 배열은 **null**을 가리킴

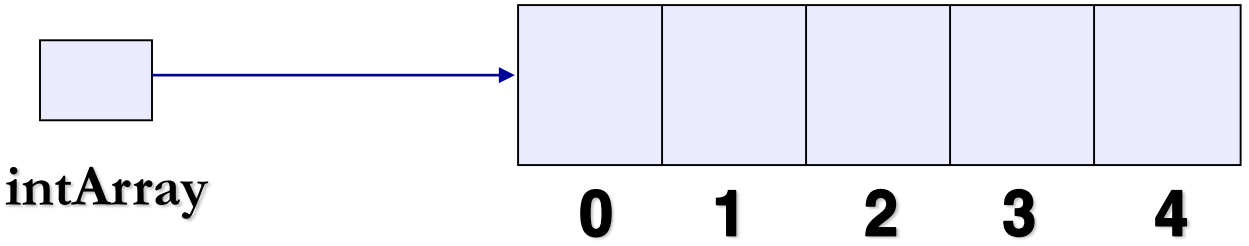
intArray = new int[5]; → 배열이 생성되면 **10**개의 **int**형 자료를 가리킴

int[] intArray = new int[5]; → 선언과 생성을 동시에

❑ 배열의 초기화

1. 생성 후 직접 입력

```
int[] intArray = new int[5]; // intArray.length의 값은 5
intArray[0] = 3;
intArray[1] = 6;
intArray[2] = 9;
```



2. 선언과 동시에 대입 가능

```
int[] intArray = {3, 6, 9, 12, 0}; // "new"는 내부적으로 호출
String[] monthName = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
   "Oct", "Nov", "Dec"};
int[] intArray = new int[] {3, 6, 9, 12, 0}; // 변형형태
```

Q) 다음 초기화의 결과는?

```
int[] intArray;
intArray = { 6, 10, 12, 0, 0 };
```

❑ 두 배열이 동일한지 비교

```
public static boolean equals(int[] a, int[] b) {
   if( a.length != b.length ) return false;
   else {
         for( int i=0; i<a.length; i++ ) {
             if( a[i] != b[i] ) return false;
         }
   }
   return true;
}
```

8. 배열 - main() 매쏘드의 매개변수

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

❑ 실행

java Echo kneel and worship Java

8. 배열 - main() 매쏘드의 매개변수 [연습문제]

```
public class Loop
{
	public static void main(String[] args)
	{
		int input = Integer.parseInt(args[0]);
		int output = 0;

		while(true)
		{
			output += input ;
			input --;
			if(input == 0) break;
		}
		System.out.println(output);
	 }
}
```

실습2. NumberTriangle.java

숫자 n을 인자(argument)로 호출했을 때, 다음과 같이 출력되는 프로그램을 작성하라.

```
1
22
333
....
.......
.........
............
nnnnnnnnnnnnnnn
```

객체지향 프로그래밍
1. 객체지향의 개요

목차

1. 객체지향이란?

❑ **객체지향 언어의 역사**

– 실세계의 모의실험(simulation)을 위해 컴퓨터를 이용, 가상세계를 구현하려는 노력으로부터 시작됨

– 1960년대 최초의 객체지향언어 Simula탄생

- 1980년대 절차방식의 프로그래밍의 한계를 객체지향방식으로 극복하려고 노력함.(C++, Smalltalk과 같은 발전된 객체지향언어가 탄생)

- 1995년 말 Java탄생. 객체지향언어가 프로그래밍 언어의 주류가 됨.

❑ **데이터의 변화/저장을 위한 알고리즘(함수) 중심으로 프로그램이 구성되는 절차지향 언어에 비해 객체지향 언어에서는 객체들의 집합이 프로그램이 되며, 객체는 데이터와 그 데이터를 처리할 수 있는 메서드를 가지게 된다.**

❑ **소프트웨어 품질 향상과 관련하여 대두된 객체지향 기술은 소프트웨어의 부품화, 소프트웨어 재사용을 주요 목표로 한다.**

❑ **요구에 맞는 객체를 만들기 위해 절차지향 언어에 비해 분석이나 설계에 더 비중 있게 생각한다.**

❑ **객체(Object), 클래스(class), 캡슐화(encapsulation), 정보은닉(information hiding), 상속(inheritance), 다형성(polymorphism) 등이 있다.**

2. 객체지향 언어의 특징

❑ 기존의 프로그래밍언어와 크게 다르지 않다.

- 기존의 프로그래밍 언어에 몇가지 규칙을 추가한 것일 뿐이다.

❑ 코드의 재사용성이 높다.

- 새로운 코드를 작성할 때 기존의 코드를 이용해서 쉽게 작성할 수 있다.

❑ 코드의 관리가 쉬워졌다.

- 코드간의 **관계**를 맺어줌으로써 보다 적은 노력으로 코드변경이 가능하다

❑ 신뢰성이 높은 프로그램의 개발을 가능하게 한다.

- 제어자와 메서드를 이용해서 **데이터를 보호**하고, 코드의 중복을 제거하여 코드의 불일치로 인한 오류를 방지 할 수 있다.

3. 객체와 클래스 - 객체

❑ 객체의 정의

- 객체는 실세계에 존재하는 것, 사물 또는 개념

- 객체는 효율적으로 정보를 관리하기 위해서 사람들이 의미를 부여하고 분류하는 논리적 단위

❑ 객체의 구성 요소

– 객체는 속성(정보)과 기능의 집합이며, 속성과 기능을 객체의 멤버(member, 구성요소)라고 한다.

– 속성은 **변수**로, 기능은 **메서드**로 정의한다. (클래스가 정의될 때)

3. 객체와 클래스 - 클래스

❑ 클래스의 정의

- 클래스는 객체를 정의해 놓은 것이다
- 클래스는 객체를 생성하는 데 사용된다.

클래스	객체
제품 설계도	**제품**
TV설계도	**TV**
붕어빵기계	**붕어빵**

3. 객체와 클래스 – 객체와 인스턴스

❑ 객체 ≒ 인스턴스

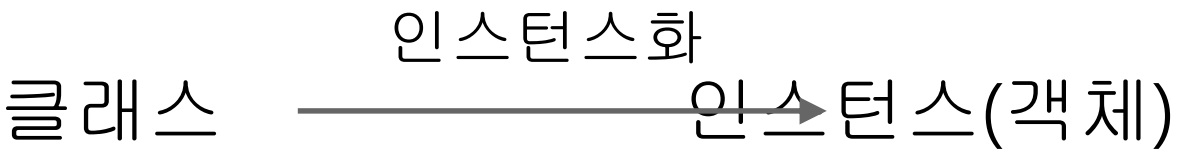
객체(object)는 인스턴스(instance)를 포함하는 일반적인 의미

❑ 인스턴스화(instantiate, 인스턴스化)

클래스로부터 인스턴스를 생성하는 것.

책상은 인스턴스다.
책상은 객체다.

책상은 책상 클래스의 객체다.
책상은 책상 클래스의 인스턴스다.



4. 객체지향 언어의 주요개념

❑ 상속

- 객체는 클래스로 부터 생성되며 생성된 객체는 속성과 메서드를 가지고 있다
- 이미 만든 객체와 비슷하지만 속성과 메서드가 약간 차이가 나는 객체를 생성해야 하는 경우?
- 기존의 클래스에서 속성과 메서드를 상속(재사용) 하고 더 필요한 속성과 메서드를 추가한다.
- 상속의 개념은 코드를 간결하게 하고 코드의 재사용성을 높이는 객체지향 개념

❑ 캡슐화(Encapsulation)

- 객체를 사용하는 쪽에서는 그 객체의 메서드를 통해 객체를 이용하고 객체의 데이터가 실제 어떻게 처리되는지 자세히 알 필요가 없다.
- 객체를 작성할 때 개발자는 숨겨야 하는 정보와 공개하는 정보를 구분하여 기술
- 객체 정보중 공개된 정보에만 접근이 가능 하다 (Information Hiding, 정보은닉)

❑ 다형성

- 객체지향의 꽃
- 메서드 오버로딩, 오버라이딩을 통해 나타난다.
- 같은 인터페이스에 다른 구현이 여러 개 가능하다는 개념

4. 객체지향 언어의 주요개념 (원칙) 정리

- **상속**
- **캡슐화(정보은닉)**
- **다형성**
  - **오버로딩(overloading) :**

    생성자, 메소드

    다중정의

    동일한 이름에 상이한 시그너처

    한 클래스 내에 선언
  - **오버라이딩(overriding) :**

    메소드

    상속과 함께 나타남 (한 메소드는 부모 클래스에 다른 메소드는 자식 클래스에 존재)

    재정의

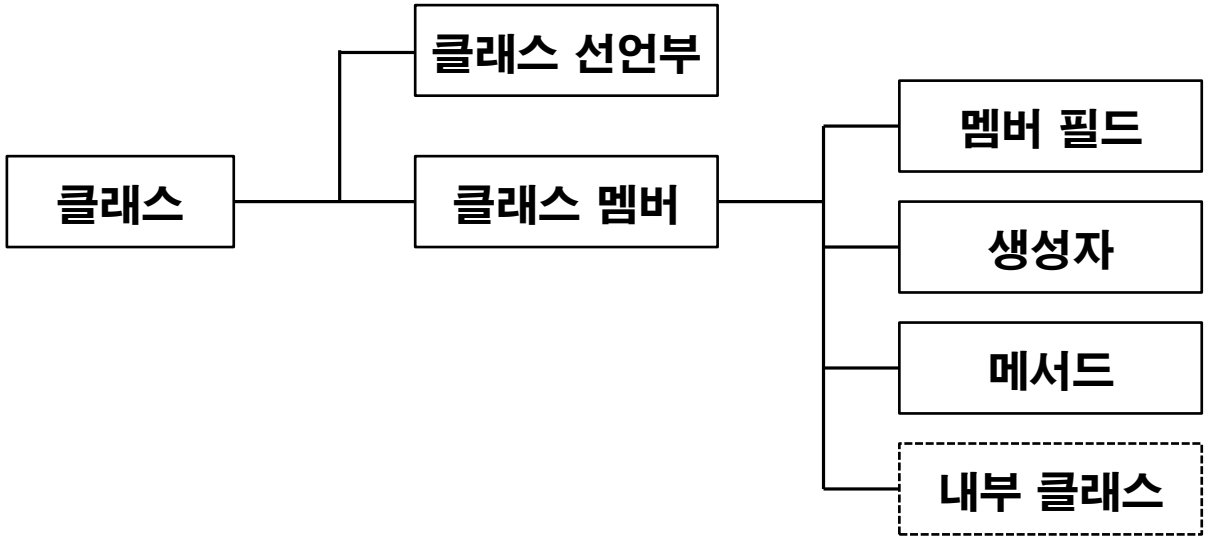
    동일한 메소드 이름에 동일한 시그너쳐를 가짐

객체지향 프로그래밍
2. 클래스와 다형성

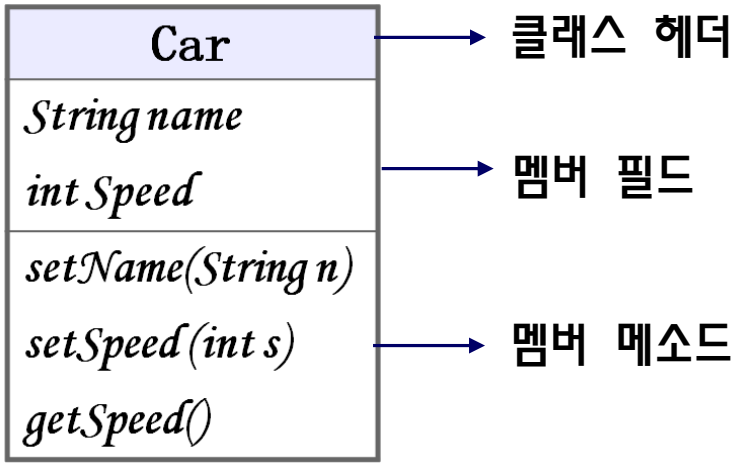
목차

1. 클래스 구조와 객체 생성

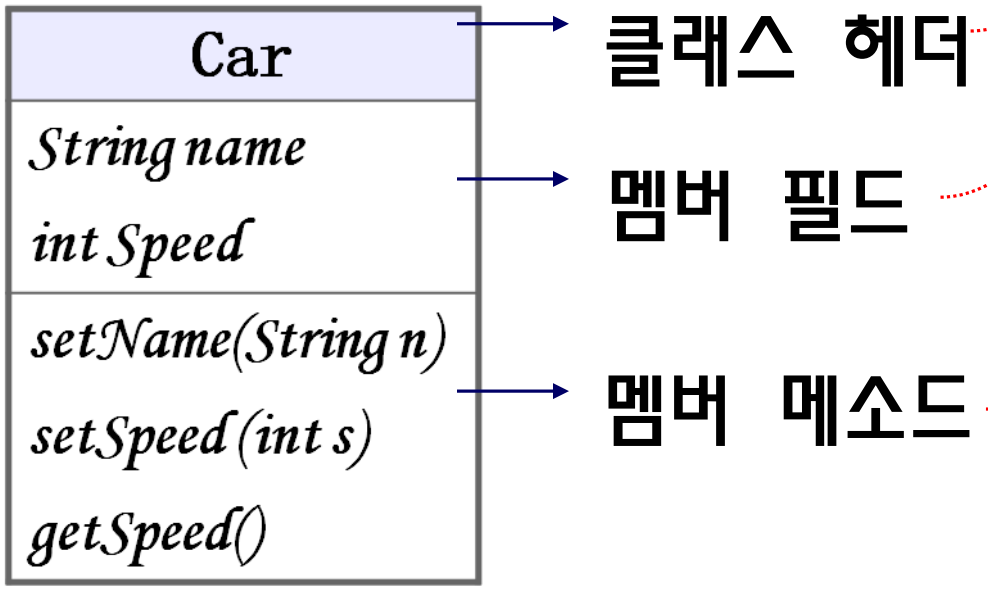
❑ 클래스의 일반 구조



❑ **Car 예제**



❑ 클래스 소스 코드




```
class Car {
    String name;
    int speed;

    public void setName(String n) {
        name = n;
    }
    public void setSpeed(int s) {
        speed = s;
    }
    public int getSpeed() {
        return speed;
    }
}
```

2. 멤버변수

❑ 선언위치에 따른 변수의 종류

"변수의 선언위치가 변수의 종류와 범위(scope)을 결정한다."

```
class Variables
{
    int iv;             // 인스턴스변수
    static int cv;      // 클래스변수(static변수, 공유변수)

    void method()
    {
        int lv = 0;     // 지역변수
    }
}
```

클래스영역

메서드영역

변수의 종류	선언위치	생성시기
클래스변수	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스변수		인스턴스 생성시
지역변수	메서드 영역	변수 선언문 수행시

2. 멤버변수

❑ 선언위치에 따른 변수의 종류 (cont'd)

▶ 인스턴스변수(instance variable)

- 각 인스턴스의 개별적인 저장공간. 인스턴스마다 다른 값 저장가능
- 인스턴스 생성 후, '참조변수.인스턴스변수명'으로 접근
- 인스턴스를 생성할 때 생성되고, 참조변수가 없을 때 가비지컬렉터에 의해자동제거됨

▶ 클래스변수(class variable)

- 같은 클래스의 모든 인스턴스들이 공유하는 변수
- 인스턴스 생성없이 '클래스이름.클래스변수명'으로 접근
- 클래스가 로딩될 때 생성되고 프로그램이 종료될 때 소멸

▶ 지역변수(local variable)

- 메서드 내에 선언되며, 메서드의 종료와 함께 소멸
- 조건문, 반복문의 블럭{} 내에 선언된 지역변수는 블럭을 벗어나면 소멸

❑ 클래스변수와 인스턴스변수

"인스턴스변수는 인스턴스가 생성될 때마다 생성되므로 인스턴스마다 각기 다른 값을 유지할 수 있지만, 클래스변수는 모든 인스턴스가 하나의 저장공간을 공유하므로 항상 공통된 값을 갖는다."

3. 생성자

❑ 생성자란?

- 인스턴스가 생성될 때마다 호출되는 '인스턴스 초기화 메서드'
- 인스턴스 변수의 초기화 또는 인스턴스 생성시 수행할 작업에 사용
- 몇가지 조건을 제외하고는 메서드와 같다.
- 모든 클래스에는 반드시 하나 이상의 생성자가 있어야 한다.

* 인스턴스 초기화 – 인스턴스 변수에 적절한 값을 저장하는 것.

```
Card c = new Card();
```

1. 연산자 new에 의해서 메모리(heap)에 Card클래스의 인스턴스가 생성된다.
2. 생성자 Card()가 호출되어 수행된다.
3. 연산자 new의 결과로, 생성된 Card인스턴스의 주소가 반환되어 참조변수 c에 저장된다.

3. 생성자

❑ 생성자의 조건

- 생성자의 이름은 클래스의 이름과 같아야 한다.

- 생성자는 리턴값이 없다. (하지만 **void**를 쓰지 않는다.)

```
클래스이름( 타입 변수명, 타입 변수명, ... ) {
    // 인스턴스 생성시 수행될 코드
    // 주로 인스턴스 변수의 초기화 코드를 적는다.
}
```

```
class Card {
    ...
    Card() {
        // 매개변수가 없는 생성자.
        // 인스턴스 초기화 작업
    }
    Card(String kind, int number) {
        // 매개변수가 있는 생성자
        // 인스턴스 초기화 작업
    }
}
```

3. 생성자

❑ 기본 생성자란?

- 매개변수가 없는 생성자

- 클래스에 생성자가 하나도 없으면 컴파일러가 기본 생성자를 추가한다.

(생성자가 하나라도 있으면 컴파일러는 기본 생성자를 추가하지 않는다.)

클래스이름() { }
Card() { } // 컴파일러에 의해 추가된 **Card**클래스의 기본 생성자**.** 내용이 없다**.**

"모든 클래스에는 반드시 하나 이상의 생성자가 있어야 한다."

4. 메소드(Method)

❑ 메소드

- 객체의 행동을 정의하는 procedure or function
- 적용대상: 특정 객체(object) or 전체 클래스(class)
- 구문:
  - ***public return_type methodName(0+ parameters){..}***
- 호출방식: “dot(.)” 연산자
  - ***objectName.methodName(arguments)***

❑ 매개변수(*parameters*)

- 메소드를 선언할 때 괄호 안에 표현된 Input 값을 나타내는 변수: **(type1 name1, type2 name2, …)**
- 메소드 호출에서 들어가는 구체적인 값은 인자(Argument)라고 함

❑ 반환값 자료형(*return_type*)

- 메소드는 0개 혹은 1개의 값을 Output으로 반환할 수 있음
- 반환값이 없을 때: **void**
- 반환값이 있을 때: **int, boolean, Car,** …
  - 반환되는 값은 메소드 선언에서 정의된 반환값의 유형과 일치해야 함

4. 메소드(Method)

메소드 선언 (with parameters)	메소드 호출 (with arguments)
public void method1(){..}	obj.method1()
public int method2(boolean b){..}	obj.method2(true)
public int method3(int x,int y,Car t){..}	obj.method3(3,4,car)

▪ *선언 예*

```
class Car {
   …
   public void setSpeed(int s) {
          speed = s;
   }
   public int getSpeed() {
          return speed;
   }
}
```

▪ *호출 예*

```
public class CarTest {
   public static void main(){
          Car mcqueen = new Car();
          …
          mcqueen.setSpeed(300);
          System.out.println(mcqueen.name + “  “ +
                 mcqueen.getSpeed());
   }
```

4. 메소드(Method) – 유형 및 예제

1. return 값과 매개변수(parameter)가 없는 메소드

```
class Car {
    String name;
    int speed;
     ….
     public void printCarInfo( ) {
          System.out.print("The Name is : " + name);
          System.out.println( "The Speed is" + speed);
     }
     .....
}
```

```
public class CarTest{
   public static void main()
    {
          Car mcqueen = new Car();
          …
          mcqueen.printCarInfo();
          …
```

2. return 값이 없고 매개변수가 있는 method

```
class Car {
   …
   public void setSpeed(int s) {
         speed = s;
   }
    …
```

```
public class CarTest {
   public static void main(){
         Car mcqueen = new Car();
         …
         mcqueen.setSpeed(300);
         …
    }
```

3. return 값과 매개변수가 있는 method

```
class intOp {
  …
  public int sum( int a, int b){
     return a+b;
  }
  …
```

```
public class intOpTest {
   public static void main(){
         intOp obj1 = new intOp();
         …
         int sum = obj1.sum(3, 9);
         …
   }
```

```
1  2   3      4
public  int  getSum ( int i, int j ) {
    int result = i + j ;   5
    return result;   6
}
```

1. 접근 지정자
2. 리턴 타입
3. 메소드 이름
4. 메소드 인자 (파라미터)
5. 구현코드
6. 리턴문

※ 메소드 시그너쳐(Signature) :
메소드 인자의 타입, 개수, 순서

실습예제1

상품 하나를 표현하는 클래스 Goods를 작성해보세요.
상품은 Goods 클래스로 표현되고 다음과 같은 4개의 필드를 가지고 있습니다.

```
String 타입의 name
int 타입의 price
int 타입의 countStock
int 타입의 countSold
```

GoodsApp 클래스의 main 메소드를 작성하여 테스트 하세요.

Goods 객체를 하나 생성하고, 이 객체에 대한 레퍼런스 변수명을 camera로 하세요
상품이름(name) 은 "Nikon",
값(price)은 400000,
재고개수(countStock)은 30
팔린개수(countSold)는 50

모든 필드는 private 으로 설정하고 getter/setter 에 의해 접근하여야 합니다.
다음 실행결과와 같게 화면에 출력하세요.

실행결과:

```
상품이름:Nikon
상품가격:400000
재고수량:30
팔린수량:50
```

실습예제2

노래를 나타내는 Song이라는 클래스를 설계하세요. Song 클래스는 다음과 같은 필드를 가지고 있습니다.

노래의 제목을 나타내는 title
가수를 나타내는 artist
노래가 속한 앨범 제목을 나타내는 album
노래의 작곡가를 나타내는 composer
노래가 발표된 연도를 나타내는 year
노래가 속한 앨범에서 트랙 번호를 나타내는 track

1) 모든 필드의 getter/setter 메소드를 작성하세요.
2) 필드는 private 으로 설정하고 getter/setter 에 의해 접근하여야 합니다.
3) 노래의 정보를 화면에 출력하는 show() 메소드를 작성하세요.
4) 아이유 의 "좋은날" 노래를 Song 객체로 생성하고 show()를 이용하여 회면에 출력하세요.

아이유 좋은날 **(Real, 2010, 3번 track,** 이민수 작곡 **)**

실습예제3

도형그리기 프로그램을 만들려고 합니다. 우선,

1. **shape(도형) 프로젝트**를 만드세요.
2. 도형의 기본이 되는 점의 위치를 나타낼 수 있는 Point 클래스를 작성하세요

   - x, y 좌표를 나타낼 수 있는 필드
   - x, y 좌표에 접근할 수 있는 getter/setter 함수
   - 다음 실행결과를 참고 해서 show() 메소드를 작성하세요.

실행결과:

```
좌표[x=2,y=5]에 점을 그렸습니다.
좌표[x=10,y=23]에 점을 그렸습니다.
```

4. 위와 같은 실행결과가 화면에 나타나도록 테스트 프로그램(ShapeTest.java)을 만드세요.

5. static의 활용

- 전역변수와 전역함수를 만들 때 사용

  모든 클래스에서 공유하는 전역 변수나 전역 함수를 만들어 사용할 수 있다.

```
public class Math {
    static int abs(int a);
    static int max(int a, int b);
}
```

- 객체를 생성하지 않고 접근할 수 있다.

- static 메소드에서는 this 사용 불가

5. static의 활용

❑ 공유 멤버를 만들고자 할 때 활용

```
class Car {
    String name;
    int speed;
    static int numberOfCars;

    public Car() {
            name = “MyCar” ;
            speed = 0;
            numberOfCars++;
    }

    public void setName(String name) {
            name = name;
    }
    public void setSpeed(int s) {
            speed = s;
    }
    public int getSpeed() {
            return speed;
    }
```

```
public class CarTest {

    public static void main()
    {
            Car mcqueen = new Car();
            System.out.println(Car.numberOfCars + “ 대의
    차가 생산되었습니다.” );
            Car hudson = new Car();
            Car marter = new Car();
            System.out.println(Car.numberOfCars + “ 대의
    차가 생산되었습니다.” );

     }
}
```

5. static의 활용

❑ **static 메소드는 static 멤버만 접근 가능하다.**

```
public class StaticMethod {
    int n;
    void f1(int x) { n = x; }
    void f2(int x) { m = x; }

    static int m;
    static void s1(int x) { n = x; }
    static void s2(int x) { f1(3); }

    static void s3(int x) { m = x; }
    static void s4(int x) { s3(3); }
}
```

실습문제 4.

다음 2개의 static 메소드를 가진 ArrayUtils 클래스를 만드세요.

ArrayUtils 클래스 이용하는 ArrayUtilsTestApp 프로그램도 만드세요

```
// int 배열을 double 배열로 변환
static double [] intToDouble( int[] source )

// double 배열을 int 배열로 변환
static int [] doubleToInt( double[] source )

// int 배열 두 개를 연결한 새로운 배열 리턴
static int [] concat( int[] s1, int[] s2 )
```

6. 오버로딩 - 생성자

❑ 기본생성자

– 기본 생성자란 인자도 없고 실행내용도 없다.

```
public class Book {
    public Book() { }
}
```

– 기본 생성자는 자동으로 생성된다.

– 기본 생성자가 자동으로 생성되지 않는 경우

생성자가 하나라도 존재하는 클래스에는 자동으로 기본 생성자가 삽입되지 않음

❑ 여러 개의 생성자를 사용할 수 있다 (생성자 오버로딩, 다형성)

실습예제5

실습예제1, 실습예제2, 실습예제3 각 클래스의 기본 생성자와 모든 필드의 초기화를 할 수 있는 생성자를 만드세요.

그리고 테스트 프로그램에서 테스트하세요.

5. 오버로딩 - this & this() & 다른 생성자 호출

❑ this

- this 예약어는 메시지를 전달받은 개체를 의미
- 현재 사용중인 객체 그 자체를 의미한다.

❑ this()

- 한 클래스 내에서 한 생성자에서 다른 생성자를 호출할 때 사용

5. 오버로딩 - this & this() & 다른 생성자 호출

```
public class Song {
   private String title;
   private String artist;
   private String album;
   private String[] composer;
   private int year;
   private int track;
   public Song() {}
   public Song( String title, String artist, String album, String[] Composer, int year, int
track) {
       this.title = title;
       this.artist = artist;
       this.album = album;
       .
       .
   }
   public Song( String title, String artist) {
     this(title, artist, “” , null, 0, 0);
   }
}
-----------------------------------------------------------------------------------------
public class SongTest {
   public static void main( String[] args ) {
       String[] composer = { “이민수”, “김이나” };
       Song song1 = new Song( “좋은날”, “아이유”, “Real”, composer, 2010, 3);
       Song song2 = new Song( “러빙유”, “씨스타”, “”, null, 0, 0 );
       Song song3 = new Song( “여수 밤바다”, “버스커 버스커”, “”, null, 0, 0 );

       Song song4 = new Song( “Like This”, “원더걸스” );
   }
}
```

5. 오버로딩 - 메소드

- 같은 클래스에 같은 이름의 메소드가 여러 개 존재할 수 있다.
- 그 메소드들은 매개변수의 데이터형이나 개수가 다른 형태로 존재하면서 구별된다. (동일한 이름에 상이한 시그너처)

※ 메소드 시그너쳐(Signature) : 메소드 인자의 타입, 개수, 순서

실습예제 6

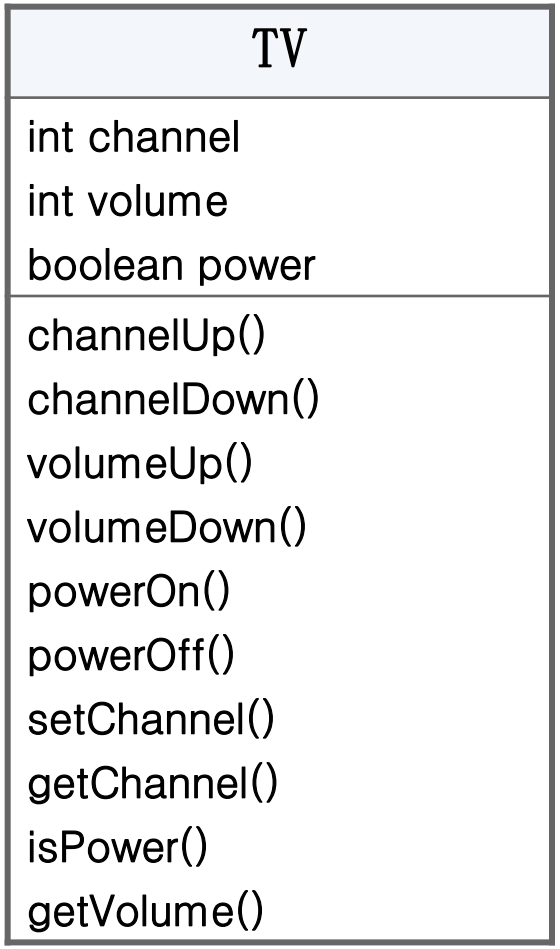
실습예제3의 Point 클래스에 점을 안보일 수 있는 기능까지 추가된 show 메소드를 하나 더 추가하고 아래 실행결과가 나도록 테스트 해보세요.

```
void show( boolean visible)
```

실행 결과:

좌표[x=2, y=5]에 점을 그렸습니다.
좌표[x=10, y=23]에 점을 그렸습니다.
좌표[x=2, y=5]에 점을 지웠습니다.
좌표[x=10, y=23]에 점을 지웠습니다.

실습예제 7




```
/**********************************************************
 * 아래 WatchTV 클래스의 main 메소드를 실행할 수 있도록,
 * 우측의 UML 그림을 참조하여 TV 클래스를 정의하십시오.
 * channel, volume, power의 초기값은 각각 7, 20, false임.
**********************************************************/
```

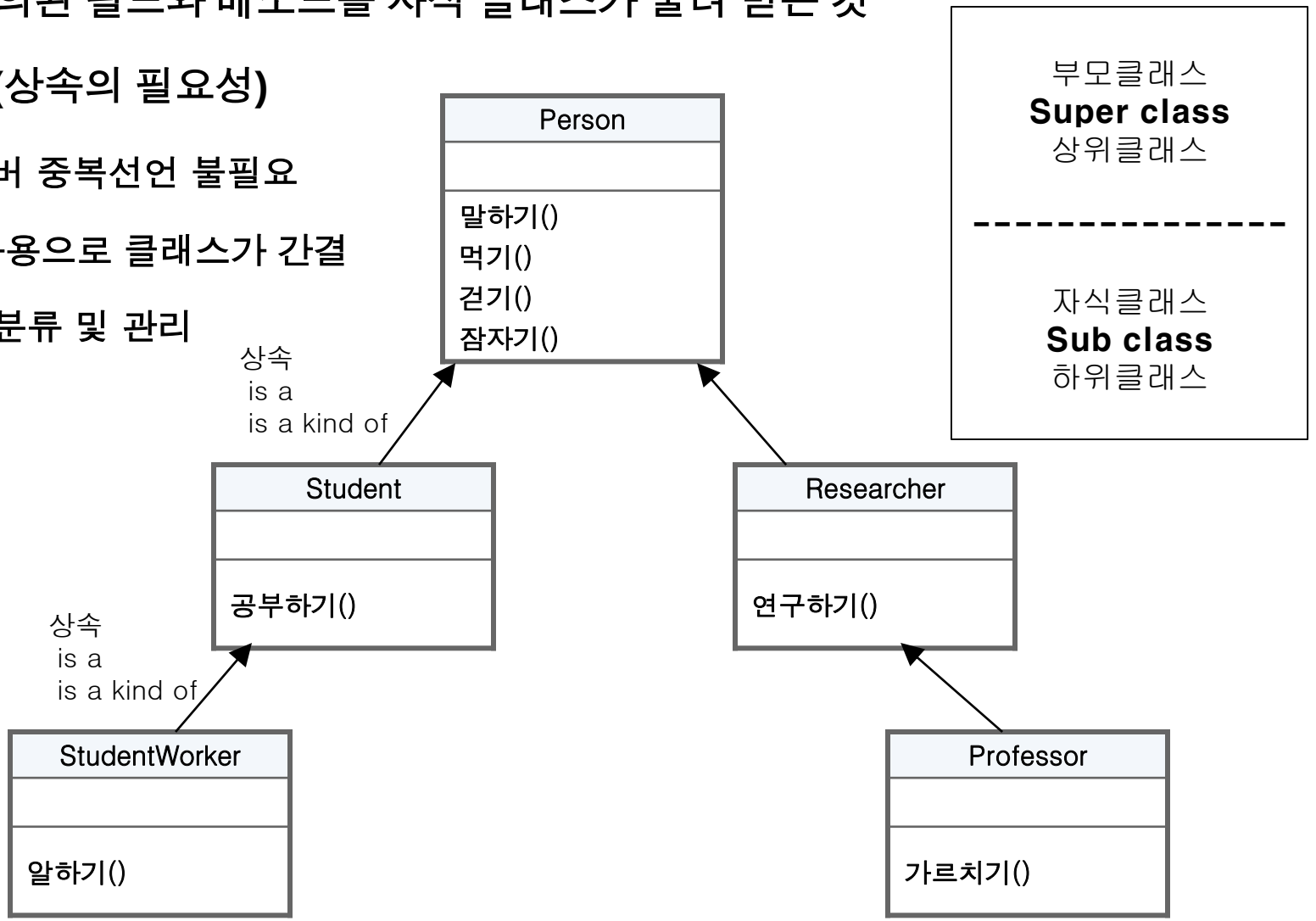
```
public class
  TV {
  ...

  }
```

객체지향 프로그래밍
3. 상속과 다형성

목차

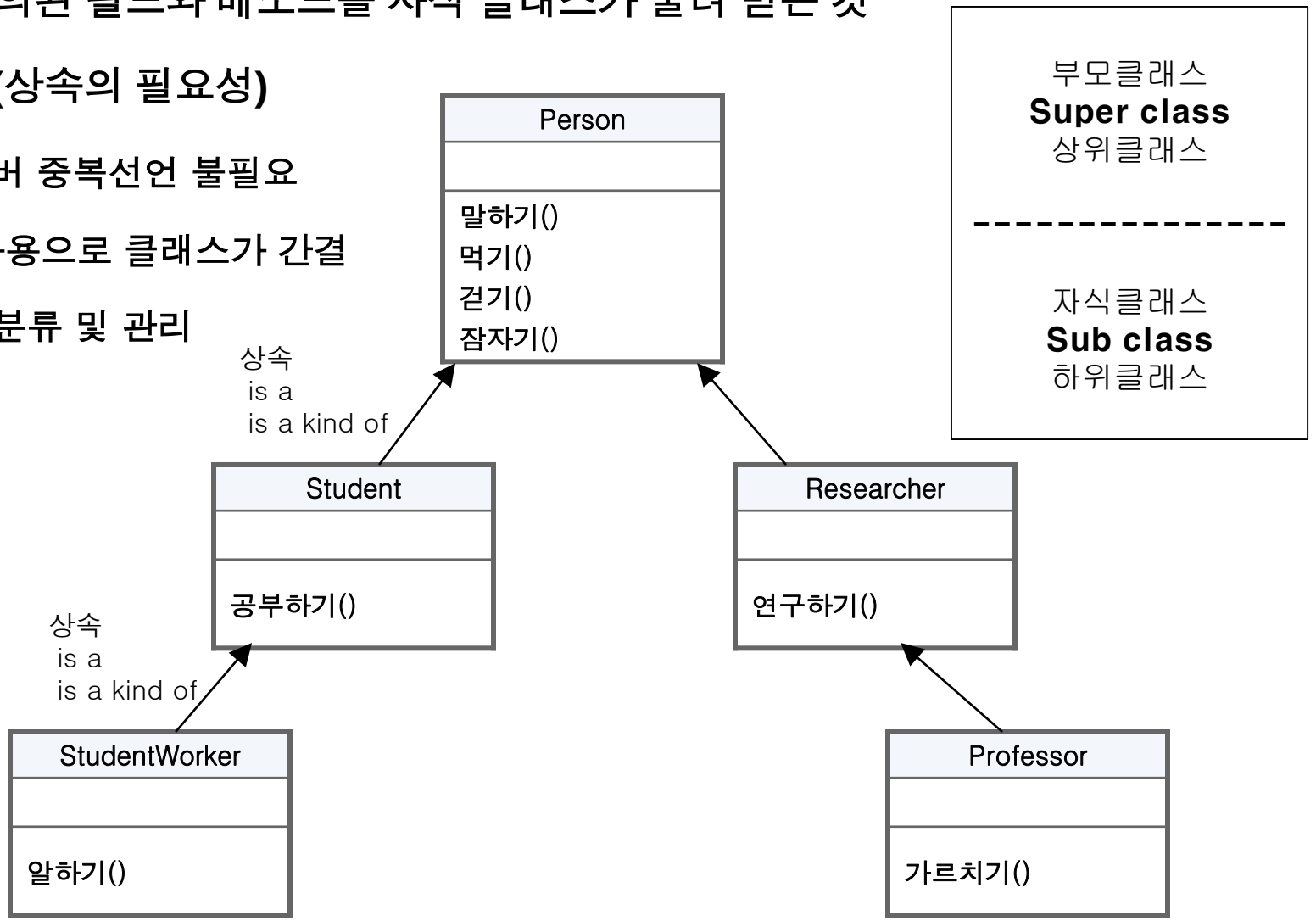
1. 상속이란?

- 부모 클래스에 정의된 필드와 메소드를 자식 클래스가 물려 받는 것
- 왜 상속을 하나? (상속의 필요성)
  - 클래스 사이의 멤버 중복선언 불필요
  - 필드, 메소드 재사용으로 클래스가 간결
  - 클래스간 계층적 분류 및 관리



1. 상속

- 부모 클래스에 정의된 필드와 메소드를 자식 클래스가 물려 받는 것
- 왜 상속을 하나? (상속의 필요성)
  - 클래스 사이의 멤버 중복선언 불필요
  - 필드, 메소드 재사용으로 클래스가 간결
  - 클래스간 계층적 분류 및 관리



1. 상속

❑ 상속선언

```
public class Person {
…
 }

 public class Student extends Person {
 …
 }

public class StudentWorker extends Student {
 …
 }
```

❑ 자바 상속의 특징

– 자바에서는 다중 상속을 지원하지 않는다.

– 자바에서는 상속의 횟수에 제한을 두지 않는다.

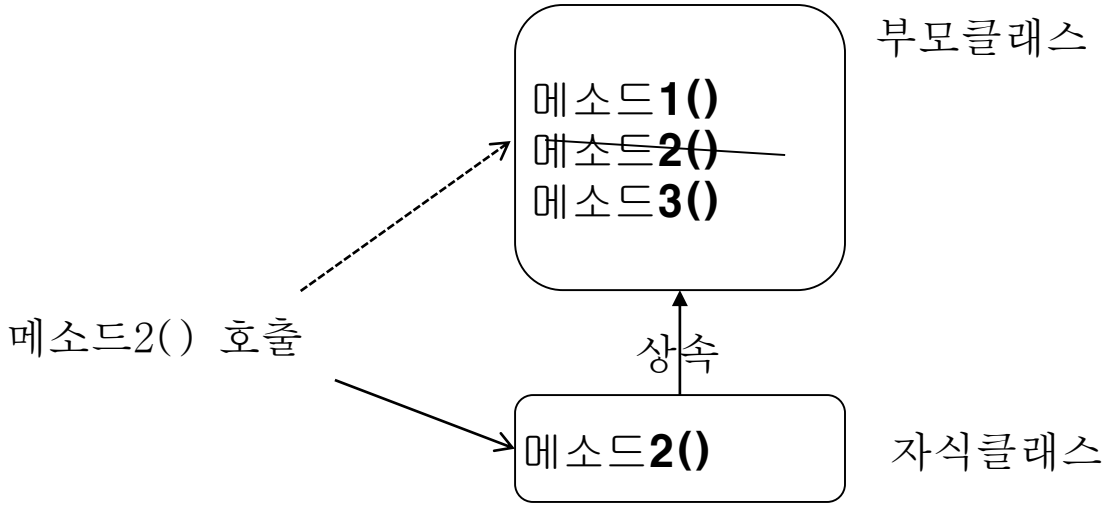
– 자바에서 계측구조의 최상위에 있는 클래스는 java.lang.Object 이다.

2. 상속과 변수

```
public class Person {
   int age;
   public String name;
   protected int height;
   private int weight;
}

-------------------------------------------------------------------------------

public class Student extends Person {
   public void set() {
      age = 30;
      name = “홍길동” ;
      height = 175;
      setHeight(99);
   }

   public static void main( String[] args ) {
      Student s = new Student();
      s.set();
   }
}
```

3. 메서드 오버라이딩

- OOP의 꽃
- 부모클래스와 자식 클래스의 메소드 사이에서 발생하는 관계
- 부모클래스의 메소드를 동일한 이름으로 재작성 (같은 이름, 같은 리턴 타입, 같은 파라미터)
- 부모클래스 메소드 무시하기



실습예제1

실습예제 2-3 Point 클래스를 상속받아 ColorPoint 클래스를 생성하려고 합니다.

1. **자식 클래스 ColorPoint에 새로운 필드 color를 추가하세요.**
2. 다음 테스트 프로그램을 작성하고 출력결과가 예시와 같도록 ColorPoint클래스의 생성자를 만들고 show 메소드도 재정의(오버라이딩) 하세요.

```
public Class ShapeTest {
   public static void main( String[] args) {
      Point a, b;

      a = new Point(2, 3);
      b = new ColorPoint(3, 4, "red");
      a.show();
      b.show();
   }
}
```

실행결과:

```
좌표[x=2,y=3]에 점을 그렸습니다.
좌표[x=3,y=4,color=red]에 점을 그렸습니다.
```

실습예제2

Shape 클래스 각각의 자식클래스 Rect, Circle를 만들어 보세요.
부모 클래스의 draw() 매소드를 자식 클래스에서 재정의(오버라이딩) 하세요.

```
public Class Shape {
   public void draw() {
       System.out.print( “구체적인 도형을 그릴 수 없습니다. 상속해서 재정의하세요”);
   }
}
```

```
public Class ShapeTest {
   public static void main( String[] args) {
      Shape rect = new Rect();
      Shape circle = new Circle();

      rect.draw();
      circle.draw();
   }
}
```

실행결과:

사각형을 그렸습니다.
원을 그렸습니다.

4. 상속과 생성자

- 자식클래스의 인스탄스가 생성될 때 자식, 부모의 생성자가 모두 실행 되는가?

- 모두 실행된다면 그 순서는?

실습예제 3

다음 부모, 자식 클래스의 기본 생성자의 출력을 통해 부모, 자식 클래스간의 생성자 실행 여부와 그 순서를 확인해 보세요.

```
public Class  Person {
   public Person() {
      System.out.print( “Person 클래스의 생성자가 호출 되었습니다.”);
   }
}

_____________________________________________________
public Class  Student extends Person {
   public Student() {
      System.out.print( “Stident 클래스의 생성자가 호출 되었습니다.”);
   }
}
```

```
public Class  Test{
   public static void main( String[] args) {

       Person student = new Student();
   }
}
```

실행결과:

?

4. 상속과 생성자 - 부모, 자식 클래스의 생성자 짝 맞추기

❑ 부모클래스의 기본 생성자 호출

특별한 지시가 없으면 자식 클래스의 생성자 가 기본 생성자이든 매개변수를 가진 생성자 이든 부모 클래스의 기본 생성자가 선택되게 된다.

❑ **super()**를 이용한 명시적 부모 클래스 생성자 호출

부모 클래스의 특정 생성자를 호출해야 할때는 super()를 이용하여 명시적으로 부모클래스의 생성자를 호출한다.

실습예제 4

실습예제3에서 Point, ColorPoint 클래스의 생성자에 출력문을 넣어 ColorPoint가 생성될 때 Point의 기본 생성자가 호출되는 여부를 확인해 보세요.

4. 상속과 생성자 - 부모, 자식 클래스의 생성자 짝 맞추기

```
public Class A {
  public A(){
     System.out.println( "생성자 A" );
  }
 public A( int x ){
     System.out.println( "매개변수 생성자 A" + x );
  }

-------------------------------------------------------------------------------------

public class B extends A {
    public B() {
      System.out.println( "생성자 B" );
    }

    public B( int x ) {
       super(x);
      System.out.println( "매개변수 생성자 B" + x );
    }
}

-------------------------------------------------------------------------------------

public class Test {
  public static void main( String[] args ) {
        B b = new B(5);
    }
}
```

5. 객체의 형변환 - 업캐스팅 & 다운캐스팅

- 업캐스팅

  자식클래스가 부모클래스 타입으로 변환되는 것

  명시적으로 타입 변환을 하지 않아도 된다.

- 다운캐스팅

  업캐스팅 된 것을 원래대로 되돌리는 것

  명시적으로 타입변환을 하여야 한다.

5. 객체의 형변환 - 업캐스팅 & 다운캐스팅 예제

```
public class Person {
   private String name;
   private String id;

   public Person( String name ) {
      this.name = name;
   }
}
-----------------------------------------------------------------------------------------
public class Student extends Person {
   private String grade;
   private String major;

   public Person( String name ) {
      super(name);
   }
}
-----------------------------------------------------------------------------------------
public class UpcastingTest{
  public static void main(String[] args){
    Person p;
    Student s = new Student( "안대혁" );
    p = s;
    System.out.println(p.name);
    p.setGrade( "A" );
  }
}
```

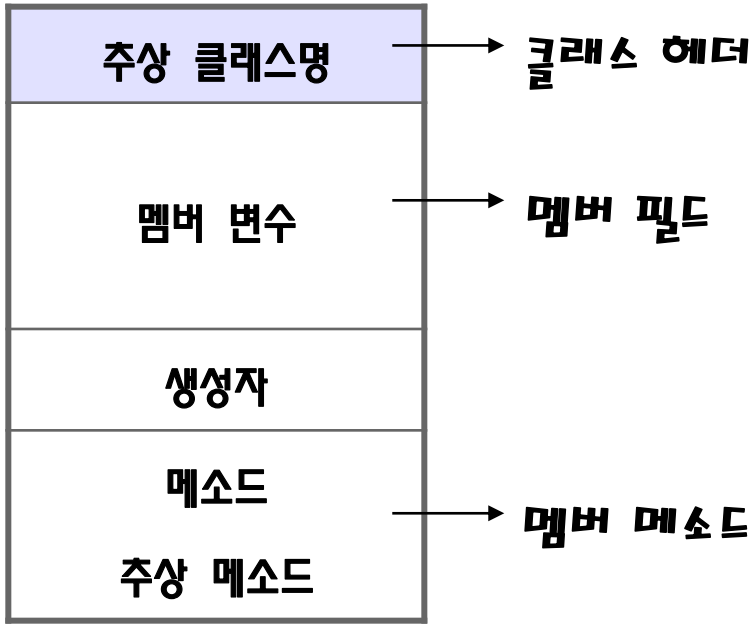
5. 객체의 형변환 - 업캐스팅 & 다운캐스팅 예제

```
public class DowncastingTest{
  public static void main(String[] args){
    Person p = new Student( “안대혁” );

    Student s;
    s = (Student)p;

    System.out.println( s.getName() );
    s.setGrade( “A” );
  }
}
```

객체지향 프로그래밍
4. 추상 클래스 & 인터페이스

목차

1. 추상 클래스



abstract Car
String name int speed boolean isSedan
Car()
setName(String n) setSpeed (int s) getSpeed() abstract boolean isSedan()

1. 추상 클래스

❑ 추상 클래스

- abstract 키워드 사용
- 확장 만을 위한 용도로 정의되는 클래스
- 하나 이상의 추상 메소드를 가짐
- 일반 메소드를 가질 수 있음
- 추상 클래스는 객체화 할 수 없음(컴파일 에러)

```
abstract class Aclass {
  public abstract void Amethod();
}

...
  Aclass a = new Aclass(); // Error!!
...
```

❑ 추상 메소드

- 추상 메소드는 메소드에 대한 구현을 가지지 않음
- 추상 클래스의 자식 클래스가 해당 메소드를 구현하도록 강요하기 위함
- 추상 메소드는 추상 클래스에만 존재할 수 있음

1. 추상 클래스

❑ 추상클래스의 상속

- 추상 클래스의 상속에도 **extends** 키워드 사용
- 추상 클래스를 상속하는 클래스는 반드시 추상 클래스의 추상 메소드를 구현해야 함
- 추상 클래스간의 상속에서는 추상클래스를 구현하지 않아도 됨

```
abstract class Aclass {
   public abstract void Amethod(); //추상 메소드
}
abstract class Bclass extends Aclass { //추상 클래스를 상속받은 추상클래스
   public abstract void Bmethod();
}
public class Cclass extends Bclass {
   public void Amethod() { … } //반드시 구현해야 함
   public void Bmethod() { … } //반드시 구현해야 함
}
```

❑ 추상클래스의 활용

- 여러 클래스들이 상당수 공통점을 가지고 있으나 부분적으로 그 처리 방식이 다를 경우 부모 클래스를 추상 클래스로 정의하여 자식 클래스들이 각각 해당 메소드를 구현

```
abstract class Shape {
   public abstract double calculateArea();
}
class Circle extends Shape {
   public double calculateArea() {
         return (double)r*r*Math.PI;
   }
}
class Rectangle extends Shape {
   public double calculateArea() {
         return (double)w*h;
   }
}
```

추상 메소드

추상 메소드의 Circle 클래스를 위한 구현

추상 메소드의 Rectangle 클래스를 위한 구현

1. 추상 클래스 – 예제

```
public abstract class List {
  protected int size;
  public int length() {
    return size;
  }
  public abstract void
    insertFront(Object item);
}
List myList;    // 문제없음
myList = new List();// 에러
```

```
public class SList extends List {
  protected SListNode head;
  public void insertFront(Object item){
      head = new SListNode(item, head);
      size++;
  }
}

List myList = new SList(); // 문제없음
myList.insertFront(obj);   // 문제없음
```

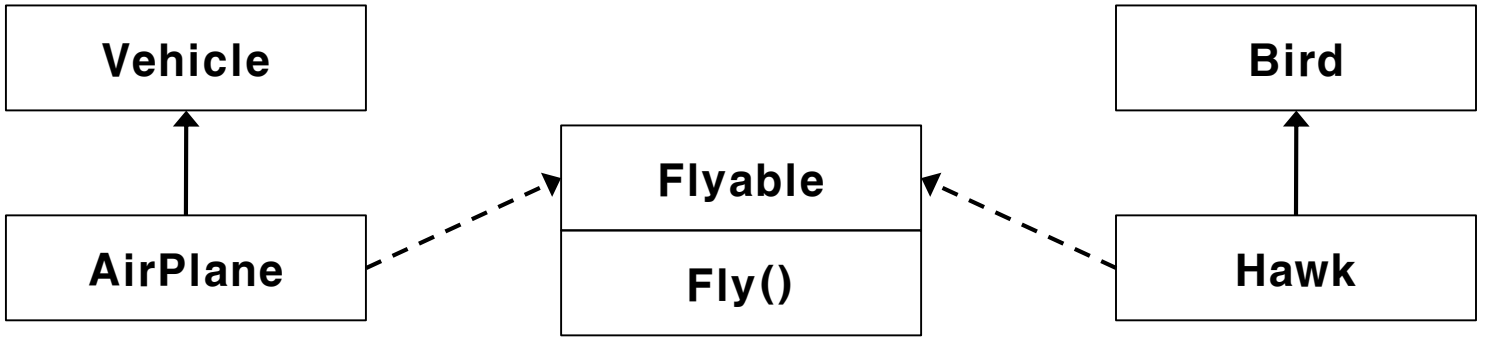
실습예제 1

- **Shape 클래스를 상속받은 Circle, Triangle, Rectangle 클래스를 정의**
  - Shape 클래스는 추상 클래스로 추상 메소드 calculateArea를 가짐
  - Shape 클래스를 상속 받은 각 클래스들은 각각 자신만의 calculateArea 메소드를 구현

2. 자바 인터페이스 (Java Interface)

❑ 자바 인터페이스(Java Interface; 이하 인터페이스)

- 개념
  - 서로 관계가 없는 물체들이 상호 작용을 하기 위해서 사용하는 장치나 시스템
  - 클래스 구조상의 관계와 상관 없이 클래스들에 의해 구현되어질 수 있는 규약
- 목적
  - 클래스들 사이의 유사한 특성을 부자연스러운 상속 관계를 설정하지 않고 얻어냄
- 활용
  - 하나 또는 그 이상의 클래스들에서 똑같이 구현되어질 법한 메소드를 선언하는 경우
  - 클래스 자체를 드러내지 않고 객체의 프로그래밍 인터페이스를 제공하는 경우



2. 자바 인터페이스 (Java Interface)

❑ 클래스 vs. 인터페이스

	일반클래스	추상클래스	인터페이스
메소드 (Method)	모두 완결한 메소드여야 함*	완결한 메소드와 추상 메소드 혼재 가능	추상 메소드, default 메소드
멤버변수 (Instance Variable)	가질 수 있음	가질 수 있음	가질 수 없음
객체화 (Instantiation)	가능	불가	불가

❑ 완결한 메소드 (Concrete method)

- 메소드의 구현을 포함한 일반적인 메소드를 concrete method라 함.
- 추상 메소드(Abstract Method)의 반대 개념

2. 자바 인터페이스 (Java Interface)

❑ 인터페이스의 상속

- 인터페이스는 다중 상속을 지원하지 않는 자바에서 다중 상속의 장점을 활용하기 위해 도입
- 한 클래스는 하나의 부모클래스를 가지며 하나 이상의 인터페이스를 구현할 수 있음
- 인터페이스 사이에서는 다중 상속이 가능
- 인터페이스는 **interface** 키워드로 선언하고 **implements** 키워드로 사용
- 인터페이스를 구현한 클래스에서 추상 메소드를 반드시 정의해야 함

```
public interface Drivable extends Steerable, Acceleratable {
}


public class Sedan extends Car implements Drivable, Breakable {
}
```

2. 자바 인터페이스 (Java Interface) – 예제

```
interface Drawable {
   public void draw(); //abstract 키워드로 지정하지 않더라도 자동으로 추상화
}

abstract class Shape {
    public abstract double calculateArea();
}

class Circle extends Shape implements Drawable {
   public double calculateArea() {
         return (double)r*r*Math.PI;
   }
   public void draw() {
         System.out.println("이 객체는 원 입니다.");
   }
}
```

2. 자바 인터페이스 (Java Interface) - default메소드

❑ default 메소드

- 인터페이스가 default키워드로 선언되면 메소드가 구현될 수 있다. 또한 이를 구현하는 클래스는 default메소드를 오버라이딩 할 수 있다.
- 인터페이스가 변경이 되면, 인터페이스를 구현하는 모든 클래스들이 해당 메소드를 구현해야 하는 문제가 있다. 이런 문제를 해결하기 위하여 인터페이스에 메소드를 구현해 놓을 수 있도록 하였다.

```
public interface Calculator {
    public int plus(int i, int j);
    public int multiple(int i, int j);
    default int exec(int i, int j){      //default로 선언함으로 메소드를 구현할 수 있다.
       return i + j;
    }
}
```

2. 자바 인터페이스 (Java Interface) - default메소드

```
public interface Calculator {
        public int plus(int i, int j);
        public int multiple(int i, int j);
        default int exec(int i, int j){      //default로 선언함으로 메소드를 구현할 수 있다.
            return i + j;
        }
}
 public class MyCalculator implements Calculator {
        @Override
        public int plus(int i, int j) {
            return i + j;
        }
        @Override
        public int multiple(int i, int j) {
            return i * j;
        }
    }
```

2. 자바 인터페이스 (Java Interface) - default메소드

```
public class MyCalculatorExam {

    public static void main(String[] args){

        Calculator cal = new MyCalculator();

        int value = cal.exec(5, 10);

        System.out.println(value);

    }

}
```

❑ 인터페이스를 구현한 객체에 대한 instanceof 연산자의 동작

```
> Circle c = new Circle();
> c instanceof Circle
true
> c instanceof Drawable
true
> c instanceof Rectangle
false
> c = new Rectangle();
> c instanceof Shape
true
```

실습 예제 2

❑ 실습 10-1의 구현에 **Drawable** 인터페이스 추가

```
public class ShapeTest {
   public static void main(String[] args) {
          Circle c = new Circle();
          Shape s = c;
          Drawable d = c;

          System.out.println(s.calculateArea());
          d.draw();
   }
}
```

객체지향 프로그래밍
5. 자바패키지

목차

1. 개요

❑ Java 패키지(Package)

- 서로 관련있는 클래스 또는 인터페이스들의 묶음
- 장점
  - 클래스들을 묶음 단위로 제공하며 필요할 때만 사용 가능(import)
  - 클래스 이름의 혼란을 막아서 충돌을 방지
  - 패키지 단위의 접근 권한 지정 가능(package)
- 사용법
  - import 패키지이름.클래스이름;

```
import java.applet.Applet;
```

  - import 패키지이름.*;

```
import java.io.*;
```

1. 개요 – 예제

```
import java.io.*;

public class Echo
{
  public static void main(String[] args) throws IOException
  {
        InputStreamReader input = new InputStreamReader(System.in);
        BufferedReader reader = new BufferedReader(input);
        String keyboardinput = reader.readLine();

        System.out.print(keyboardinput);
        System.out.println(" has entered.");
  }
}
```

2. 패키지 적용

❑ 새로운 패키지 작성

- 패키지의 이름과 계층 구조를 결정
- 패키지를 위치시킬 디렉토리에 패키지의 계층구조와 동일한 디렉토리 구조 생성
- 패키지에 추가할 클래스들을 생성하고 해당 디렉토리로 이동
- 새로 생성된 패키지가 위치한 디렉토리를 환경변수에 추가(CLASSPATH)

```
package kr.co.sunnyvale;
public class Sunnyvale {
    public Sunnyvale() {}
    public void greeting(String name) {
            System.out.println( “Hello,
      “+name);
    }
}
```

- C:\MyPackage\kr\co\samsung\Sunnyvale.class
- CLASSPATH 에 C:\MyPackage 추가

```
import kr.co.sunnyvale.*;
public class MyPackageTest {
    public static void main(String[] args) {
            Sunnyvale  sunny = new Sunnyvale();
            sunny.greeting( “Sunnyvale” );
    }
}
```

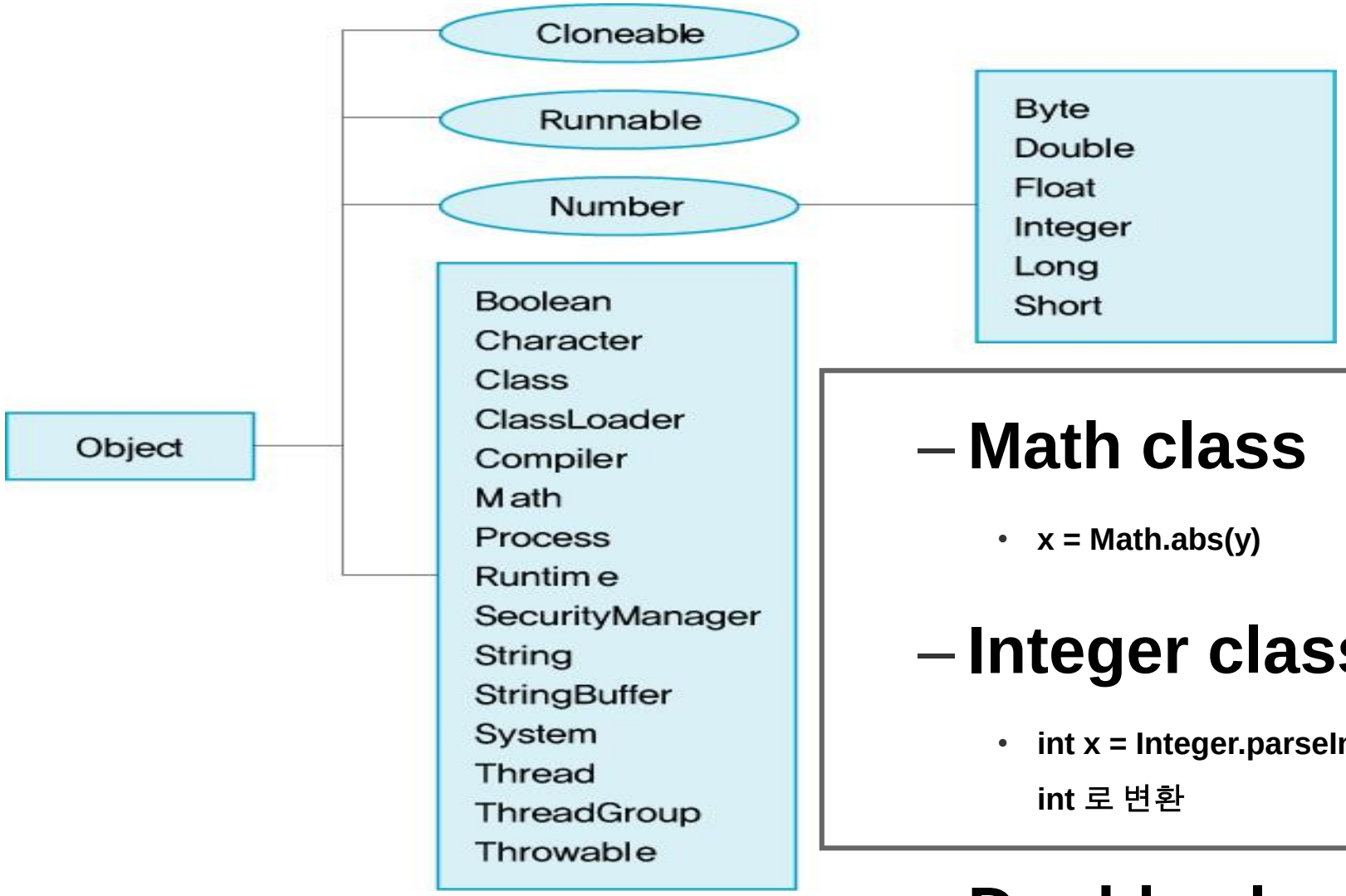
3. Java API Package

❑ Java API

- JDK에 정의된 자바 표준 패키지
- 프로그래머가 새로운 클래스를 정의하여 사용하는 것 보다 자바API에서 제공하는 많은 클래스들을 재사용하여 쉽게 프로그램 작성 가능

자바 API	제공하는 기능
java.lang.*	**명시적으로 지정하지 않아도 자동으로 import 되는 패키지. 자바 프로그램이 기본적으로 필요로 하는 클래스와 인터페이스 포함**
java.io.*	**데이터를 입력받고 출력할 수 있도록 하는 클래스 포함**
java.net.*	**네트워크를 통하여 통신할 수 있도록 해주는 클래스 포함**
java.util.*	**날짜/시간 조작, 난수 발생 등 각종 유틸리티 클래스와 인터페이스 포함**

3. Java API Package – java.lang 패키지



– Math class

- x = Math.abs(y)

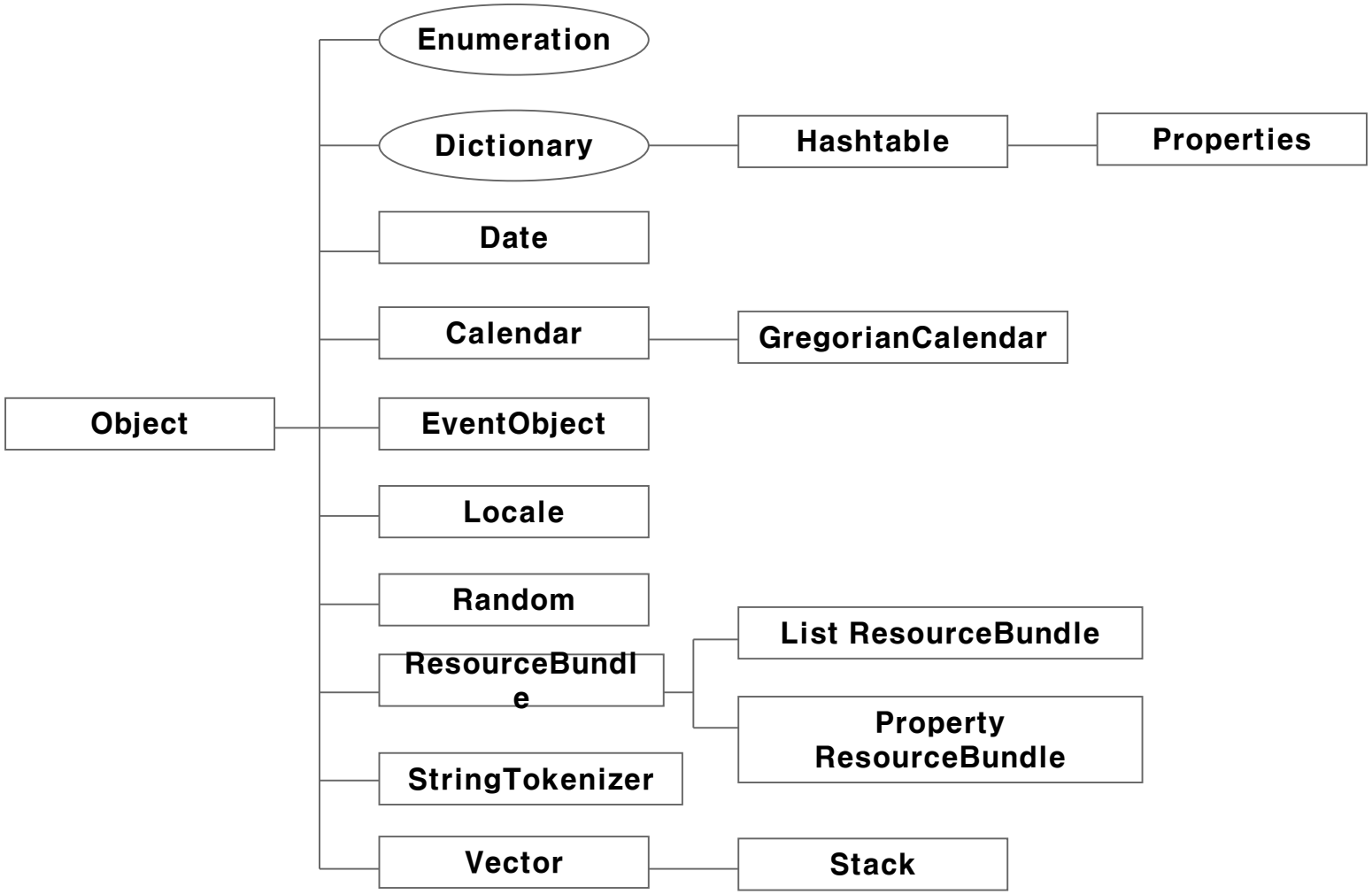
– Integer class

- int x = Integer.parseInt("1984"); // 문자열을 int 로 변환

– Double class

- double d = Double.parseDouble("3.14");

3. Java API Package) – java.util 패키지



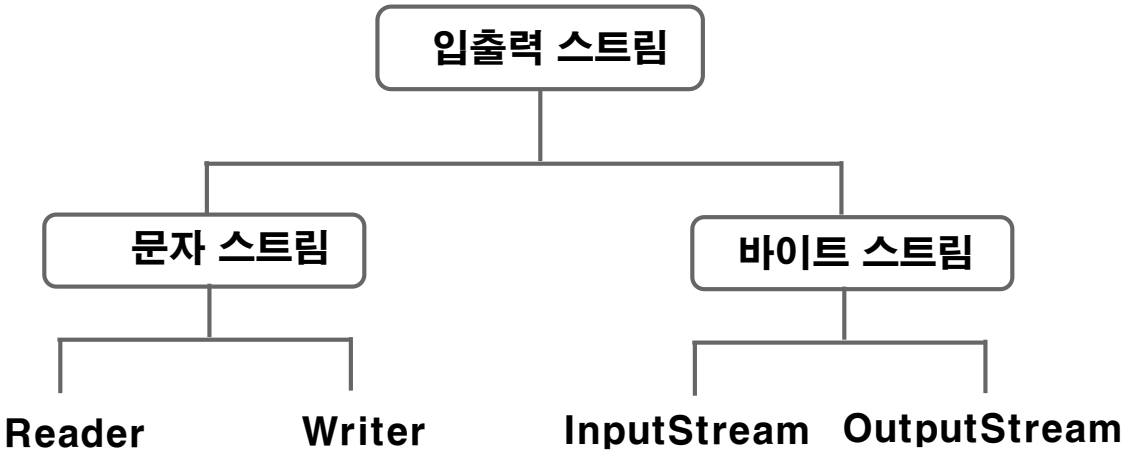
3. Java API Package – java.io 패키지

❑ 문자 스트림

– 16 Bit로 코드화 된 문자나 문자열을 읽거나 쓰는 데 이용(unicode)

❑ 바이트 스트림

– 숫자나 binary 자료를 읽거나 쓰는 데 이용



객체지향 프로그래밍
6. 예외처리

목차

1. 예외처리(Exception handling) - 개념

❑ 예외(Exception)

- 프로그램이 실행되는 동안 발생할 수 있는 비정상적인 조건
- 번역시의 에러가 아닌 실행시의 에러를 예외라 함

❑ 자바에서의 예외처리

- 예외처리를 위한 Exception 클래스 정의
- 기본적인 예외는 자바에 미리 정의된 예외를 통해 처리 가능
- 사용자가 필요한 예외를 직접 정의할 수 있음
- 예상되는 예외는 미리 처리해주면 무조건적인 프로그램의 종료를 피할 수 있음
- 예외처리의 사용은 프로그램의 신뢰성을 높여줌

1. 예외처리 - 예제

```
public class ExceptionTest {
  public static void main(String[] args) {
        int a = 0;
        double b;
        b = 100/a; //java.lang.ArithmeticException 발생
        System.out.println("Some more codes"); //예외 발생으로 수행되지 않음
  }
}
```

2. 예외처리 방법

❑ try-catch-finally 구문 이용

0

```
try {
    1  예외가 발생할 가능성이 있는 실행문
} catch ( 처리할 예외 타입  선언  ) {
    2  예외 처리문
} finally {
    3  예외 발생 여부와 상관없이 무조건 실행되는 문장  ( 생략가능 )
}
```

4

❑ Try 블록에서 예외가 발생한 경우 : 0 -> 1 -> 2-> 3->4

❑ Try 블록에서 예외가 발생하지 않은 경우 : 0 -> 1 -> 3 -> 4

2. 예외처리 방법 - 예제

```
public class ExceptionTest {
   public static void main(String[] args) {
         int a = 0;
         double b;
         try {
             b = 100/a;
             System.out.println("Some more codes in try block");
         }
         catch(ArithmeticException e) {
             System.out.println("Exception occurred : "+e);
         }
         catch(IOException e2) {
             System.out.println("One more catch block");
         }
         finally {
          System.out.println("Some more codes in finally block");
         }
   }
}
```

실습문제1

```
public class ArrayException {
   public static void main (String[] args) {
      int[] intArray = new int[5];
      intArray[0] = 0;
      for (int i = 0; i < 5; i++) {
        intArray[i+1] = i+1 + intArray[i]; // i=4인 경우 예외 발생
        System.out.println("intArray["+i+"]"+"="+intArray[i]);
      }
    }
}
```

예외가 발생하면 예외처리를 하세요.

3. 예외 발생의 예와 종류(1)

❑ 예외의 구분

- Checked Exception : 컴파일 할 때 확인 되는 예외로 예외처리가 필요함
- Unchecked Exception : 실행시점에 확인되는 예외로 예외처리를 하지 않아도 컴파일 됨

❑ 예외처리가 유용한 경우

- 배열과 연관된 for 반복문
  - 배열 참조값이 배열의 크기를 벗어난 경우 예외 발생
- 파일을 다루는 경우
  - 해당 파일이 존재하지 않거나
  - 다른 프로세스에 의해 사용중인 경우 예외 발생
- 입출력을 다루는 경우
  - 이미 닫힌 입출력 스트림에 대해 작업하려 할 경우 예외 발생

3. 예외 발생의 예와 종류(2)

예외의 종류	발생하는 경우
ArithmeticException	0으로 나누거나 0으로 나눈 나머지를 구하려 할 경우 발생하는 예외
NullPointerException	객체가 할당되지 않은 레퍼런스를 통하여 멤버 변수나 멤버 메소드에 접근하려할 경우 발생하는 예외
IOException	올바른 입출력 동작이 아닐 경우 발생하는 예외
FileNotFoundException	읽거나 쓰고자 하는 파일이 존재하지 않거나 사용 가능하지 않은 경우 발생하는 예외
ArrayIndexOutOfBoundsException	배열의 참조가 배열의 크기를 벗어난 경우 발생하는 예외

4. 위험요소가 있는 메소드

- **메소드를 정의할 때 메소드의 내부에서 예외가 발생할 가능성이 있을 경우**
  - try/catch 문으로 예외를 직접 처리하거나
  - 해당 메소드를 호출하는 메소드에서 예외를 처리하도록 명시할 수 있음
- **throws 키워드를 사용하여 예외의 종류를 적어줌**

```
public class ThrowsText {
  public void SuspiciousMethod() throws IOException, FileNotFoundException
  {
        throw new IOException(); //강제로 예외 발생
  }
}
```

5. 사용자 정의 예외

- 예외의 최상위 클래스인 **Exception** 클래스를 상속받아 새로운 예외를 정의할 수 있음
- 일반적으로 생성자만 구현

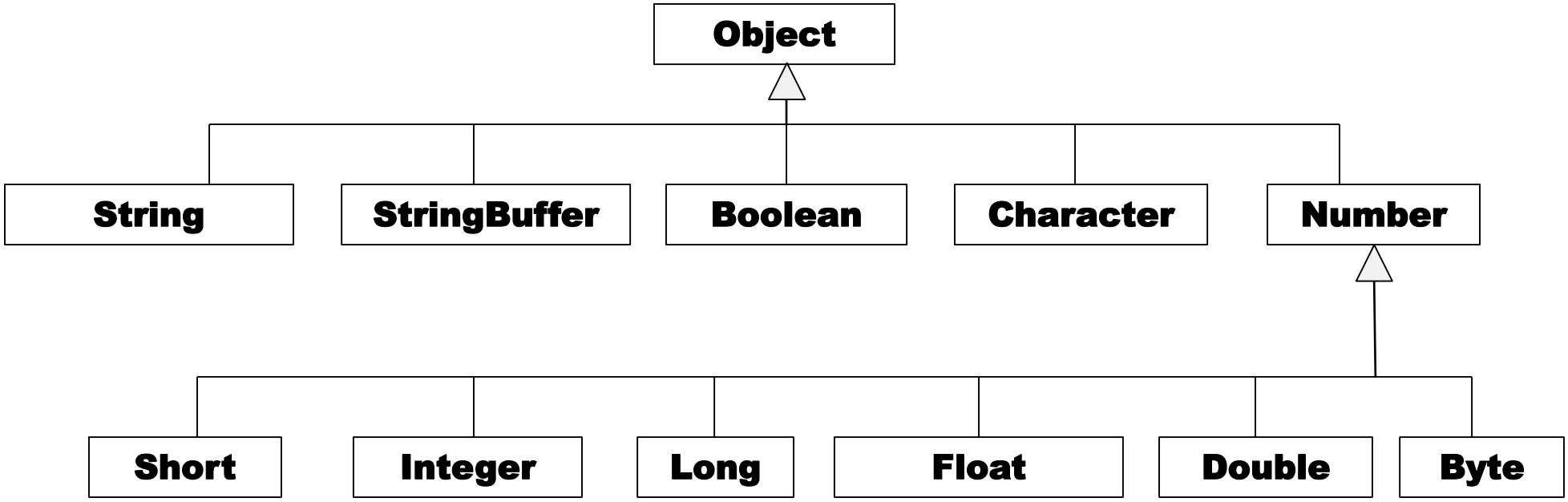
```
public class DivideByZeroException extends Exception {
  public DivideByZeroException()
  {
       super("Dividing by 0");
  }
  public DivideByZeroException(String msg)
  {
       super(msg);
  }
}
```

자바기본API
1. 언어와 시스템 API

목차

1. 개요

- 자바 프로그램에서 가장 많이 사용되는 패키지
- Import 를 사용하지 않아도 자동으로 포함



2. Object 클래스

- 모든 클래스의 최상위 클래스
- 명시적 **extends java.lang.Object** 없이도 자동으로 상속 받게된다.

```
public class Point {
   private int x;
   private int y;
   public Point( int x, int y ) {
     this.x = x;
     this.y = y;
    }
}

-------------------------------------------------------------------------------
public class LangClassTest {
  public static void main( String[] args ) {

     Point p =  Point(2, 3);

     System.out.println( p.getClass().getName() );
     System.out.println( p.hashCode() );  // 객체를 유일하게 구분할 수 있는 정수 id
     System.out.println( p.toString() );
     System.out.println( p );
  }
}
```

2. Object 클래스 - 객체를 문자열로 변환, toString() 메소드

```
public class Point {
   private int x;
   private int y;
   public Point( int x, int y ) {
     this.x = x;
     this.y = y;
    }
}

-------------------------------------------------------------------------------
public class LangClassTest {
  public static void main( String[] args ) {

     Point p =  Point(2, 3);

     System.out.println( p.getClass().getName() );
     System.out.println( p.hashCode() );  // 객체를 유일하게 구분할 수 있는 정수 id
     System.out.println( p.toString() );
     System.out.println( p );
  }
}
```

2. Object 클래스 - toString() 오버라이딩

```
public class Point {
   private int x;
   private int y;
   public Point( int x, int y ) {
     this.x = x;
     this.y = y;
   }
   public String toString(){
     resturn “Point( “ + x + “,” + y + “)”; // Point 클래스만의 toString()
   }
}

-------------------------------------------------------------------------------
```

출력결과:
Point(2,3)

2. Object 클래스 - 객체 비교 & equals() 메소드

- 두 객체의 비교시 == 와 Object 클래스의 equals() 메소드를 사용한다.
- == 와 equals() 의 상당한 차이점

```
public class Point {
   private int x;
   private int y;
   public Point( int x, int y ) {
     this.x = x;
     this.y = y;
   }
}
-------------------------------------------------------------------------------
public class LangClassTest {
  public static void main( String[] args ) {

    Point a = new Point(2, 3);
    Point b = new Point(2, 3);
    Point c = a;

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

2. Object 클래스 - 객체 비교 & equals() 메소드

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

    Point a = new Point(2, 3);
    Point b = new Point(2, 3);

    System.out.println( a == b );
    System.out.println( a.equals( b ) );

    String s1 =  new String( “hello” );
    String s2 =  new String( “hello” );

    System.out.println( s1 == s2 );
    System.out.println( s1.equals( s2 ) );
  }
}
```

2. Object 클래스 - equals() 오버라이딩

```
public class Point {
   private int x;
   private int y;
   public Point( int x, int y ) {
     this.x = x;
     this.y = y;
   }
   public boolean equlas( Point p) {
      if( x = p.x && y == p.y) {
          return true;
      } else {
          return false;
      }
   }
}
-------------------------------------------------------------------------------
public class LangClassTest {
  public static void main( String[] args ) {

    Point a = new Point(2, 3);
    Point b = new Point(2, 3);

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

실습예제 #1

Rect 클래스를 만들고 equals() 오버라이딩 하기

int 타입의 width, height의 필드를 가지는 Rect 클래스를 작성하고 두 Rect객체의 width, height 필드에 의해 구해지는 면적이 같으면 두 객체가 같은 것으로 판별하도록 equals()를 오버라이딩 하세요.

```
public class LangClassTest {

  public static void main( String[] args ) {

    Rect a = new Rect(2, 3);
    Rect b = new Rect(3, 2);
    Rect c = new Rect(3, 4);

    if(a.equals(b)) System.out.println(  “사각형 a 와 사각형 b는 같습니다.”  );
    if(a.equals(c)) System.out.println(  “사각형 a 와 사각형 c는 같습니다.”  );
    if(b.equals(c)) System.out.println(  “사각형 b 와 사각형 c는 같습니다.”  );
  }

}
```

실습예제 #2

int 타입의 x, y, radius 필드를 가지는 Circle 클래스를 작성하세요.
equals() 메소드를 재정의 하여 두 개의 Circle 객체의 반지름이 같으면 두 Circle 객체가 동일한 것으로 판별하도록 하세요.

2. String 클래스 – Character vs. String

❑ 문자(Character):

- 단순 자료형(char)
- 문자 선언 **`char letter;`**
- 문자열 할당 **`letter = 'A';`**
- 문자열 사용 **`System.out.println(letter);`** **`System.out.println((int)letter);`**

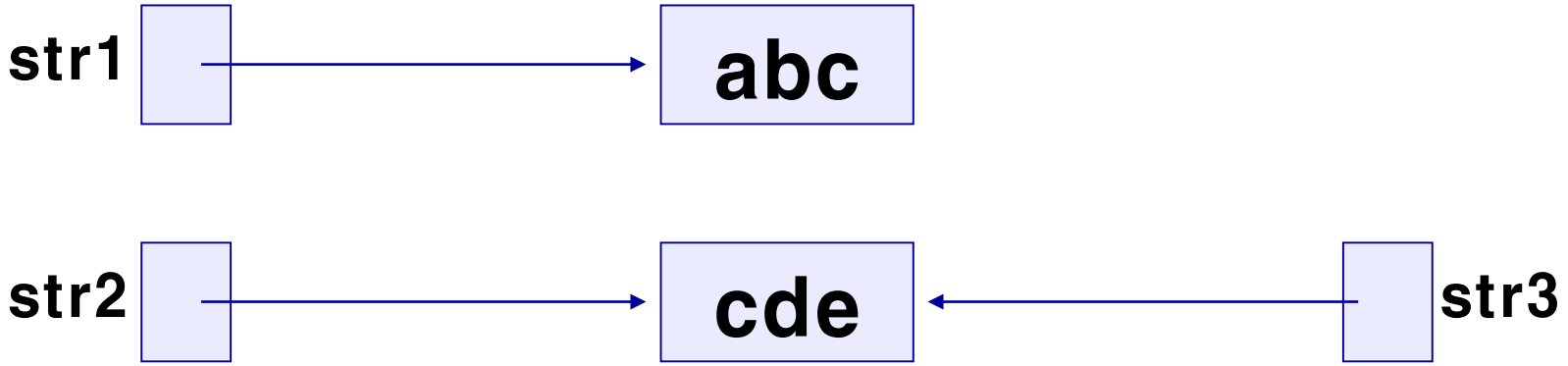
❑ 문자열(String) 클래스

- 연속된 문자(character)들을 저장하고 다루기 위한 클래스
- 문자열 상수: " "로 둘러싸인 문자열 → **한번 만들어진 String 객체는 변경 불가함(immutable)**
- 문자열 선언 **`String greeting;`**
- 문자열 할당 **`greeting = "Hello JAVA!";`**
- 문자열 사용 **`System.out.println(greeting);`**
- 특수 문자의 표현(Escape characters)
  - \' , \" , \\, \n, \r, \t

```
System.out.print("Hello "JAVA!"");
System.out.print("Hello \"JAVA!\"");
System.out.print("Hello \"JAVA!\"\n");
```

2. String 클래스 – 메모리 그림(I)

```
String str1;
String str2, str3;  // String 클래스변수 str1, str2, str3 선언
str1 = "abc"; // str1은 생성된 String 클래스의 객체(Object)를 가리킴
str2 = "cde"; // str2은 생성된 String 클래스의 객체(Object)를 가리킴
str3 = str2; // str3에 str2의 값 할당
```



2. String 클래스 – String 연산

❑ “+” 연산자

```
String greet = “Hello”;
String name = “JAVA”;
System.out.println(greet+name+”!”);
System.out.println(greet+” “+name+”!”);
```

❑ Index

- String 객체 내의 문자 인덱싱은 배열과 같이 0부터 시작됨
- charAt(position) : 해당 위치의 문자를 반환
- substring(start, end) : start부터 end까지의 문자들을 새로운 문자열로 반환

```
String greeting = “Hello JAVA!”;
greeting.charAt(0); //’H’
greeting.charAt(10);
greeting.substring(1,3); //”ell”
```

Index	0	1	2	3	4	5	6	7	8	9	10
char	H	e	l	l	o		J	A	V	A	!

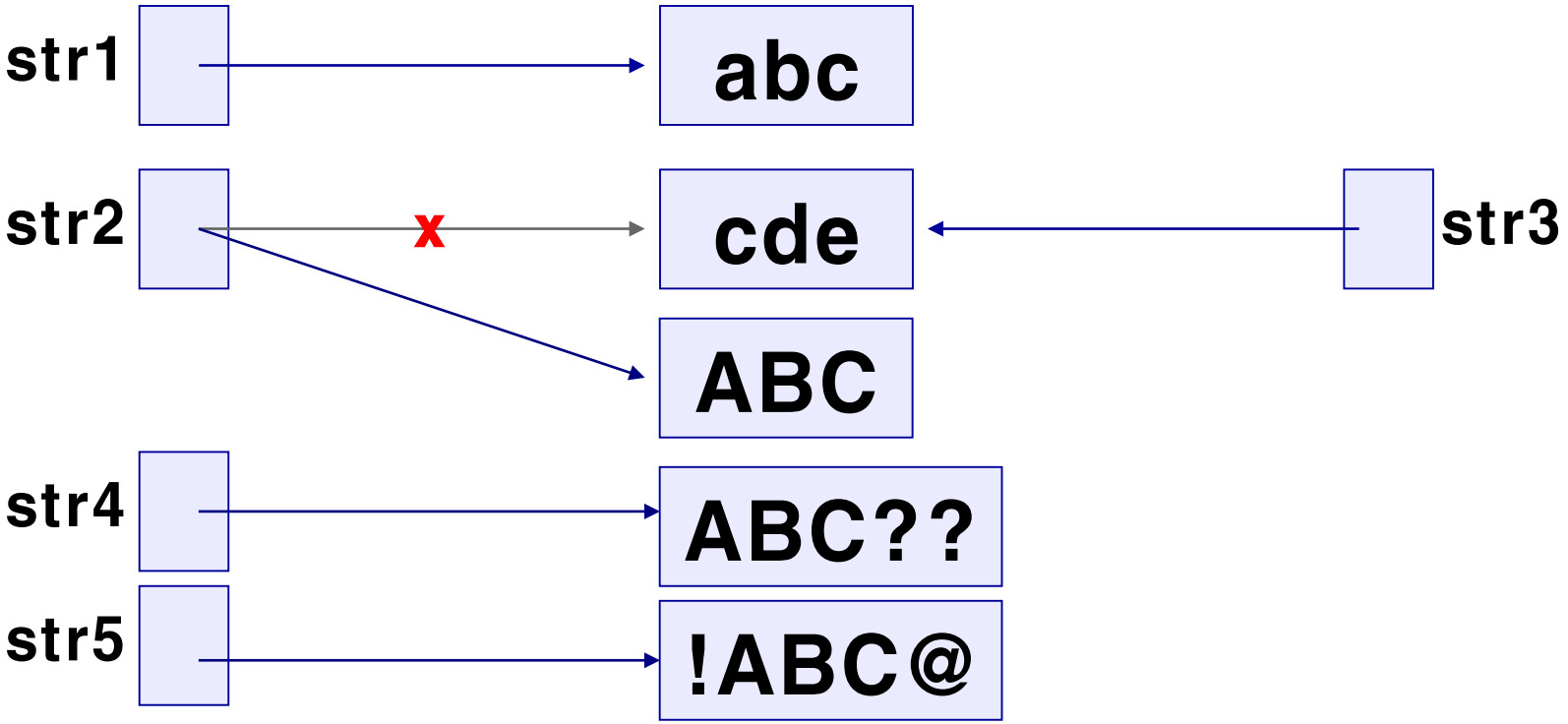
2. String 클래스 – 예제

Q) 메모리 그림과 API 문서를 활용해서 화면에 출력되는 값을 예상해 보십시오.

```
str2 = str1.toUppercase();
String str4 = str2.concat("??");
String str5 = "!".concat(str2).concat("@");
String xyz = "xyz"
System.out.println("str1: " + str1);
System.out.println("str2: " + str2);
System.out.println("str4: " + str4);
System.out.println("str5: " + str5);
System.out.println(“xyz” + xyz);

System.out.println(str1.charAt(1));
System.out.println(str1.length());
```

2. String 클래스 – 메모리 그림(II)

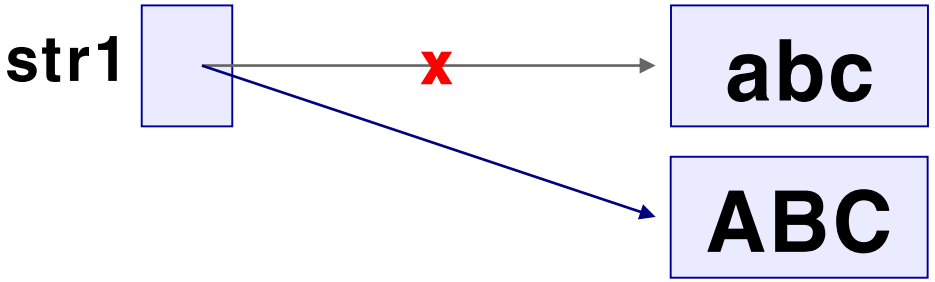
```
str2 = str1.toUppercase();
String str4 = str2.concat("??");
String str5 = "!".concat(str2).concat("@");
```



2. String 클래스 – 메모리 그림(III)

※Java의 String 객체는 한번 생성되고 나면 변경할 수 없고 완전히 새로운 String 클래스 객체가 생성됨

```
str1 = str1.toUppercase();
```



2. String 클래스 – 스트링 리터럴과 new String() 의 차이점

```
public class LangClassTest {

  public static void main( String[] args ) {

    String s1 = “hello” ;
    String s2 = “hello” ;

    String s3 =  new String( “hello” );
    String s4 =  new String( “hello” );

    System.out.println( s1 == s2 );
    System.out.println( s3 == s4 );
    System.out.println( s2 == s4 );

  }
}
```

실습예제 #3

다음의 실행결과와 같이 출력하는 프로그램을 작성하세요.

1) “aBcAbCabcABC” 문자열 String 객체를 생성한다.
2) 3번째 문자열 출력한다,
3) “abc” 문자열이 처음으로 나타나는 인덱스를 추적한다.
4) 위 문자열의 문자 개수를 출력한다.
5) ‘a’ 를 ‘R’ 로 대체한 결과를 출력한다.
6) “abc” 문자열을 “123’ 문자열로 대체한 결과를 출력한다.
7) 처음 3개의 문자열만을 출력한다.
8) 문자열을 모두 대문자로 변경하여 출력한다.

실습예제 #4. StringTest.java

```
public class StringTest {
  public static void main( String[] args ) {
    String a = new String( “ abcd” );
    String b = new String( “,efg ” );
    // 문자열 연결
    a = a.concat(b);
    System.out.println( a );
    // 공백제거
    a = a.trim();
    System.out.println( “---” + a + “---” );
    //  문자열 대치
    a = a.replace( “ab” , “12” );
    System.out.println( a );
    //  문자열 분리
    String s[] = a = a.split( “,” );
    for(int i = 0; I < s.length; i++){
       System.out.println( s[i] );
    }
    // 서브스트링
    a = a.substring(3);
    System.out.println( a );
    // 문자열의 문자
    char c = a.charAt(2);
    System.out.println(c);
  }
}
```

3. StringBuffer 클래스

- 가변크기의 버퍼를 가짐
- **String** 객체가 한 번 만들어진 문자열을 수정할 수 없는 것과는 달리 **StringBuffer** 객체는 수정 가능하다. **(** 스트링 버퍼의 크기는 문자열 길이에 따라 가변적으로 변한다**)**

```
public class SBTest {
  public static void main( String[] args ) {
    StringBuffer sb =  new StringBuffer( “This” );  //생성
    System.out.println( sb );
    sb.append( “ is pencil” ); // 문자열 덧붙이기
    System.out.println( sb );
    sb.insert(7, “ my” ); // 문자열 삽입
    System.out.println( sb );
    sb.insert(8, 10, “ your” ); // 문자열 대치
    System.out.println( sb );
    sb.setLength(5); // 스트링 버퍼 내 문자열 길이 설정
    System.out.println( sb );
  }
}
```

4. Wrapper 클래스

- 클래스가 아님
- **8개의 기본 데이터형을 객체형식으로 다루기 위한 클래스들의 통칭**

  byte -> Byte short -> Short int -> Integer long -> Long

  char -> Character

  float -> Float double -> Double

  boolean -> Boolean

- 사용하는 이유

  - 자바세상은 객체만 있기 때문에 개체를 대상으로 처리하는 경우가 많음

  - 어떤 클래스는 객체만을 다루기 때문에 기본 데이터 형을 쓸 수 없다.

4. Wrapper 클래스 - 객체 생성

```
public class LangClassTest {

  public static void main( String[] args ) {

    Integer i = new Integer( 10 );
    Character c = new Character(  'c'  );
    Float f = new Float(3.14);
    Boolean b = Boolean( true );

    Integer i = new Integer(  "10"  );
    Double d = new Double(  "3.14"  );
    Boolean b = Boolean(  "false"  );
  }
}
```

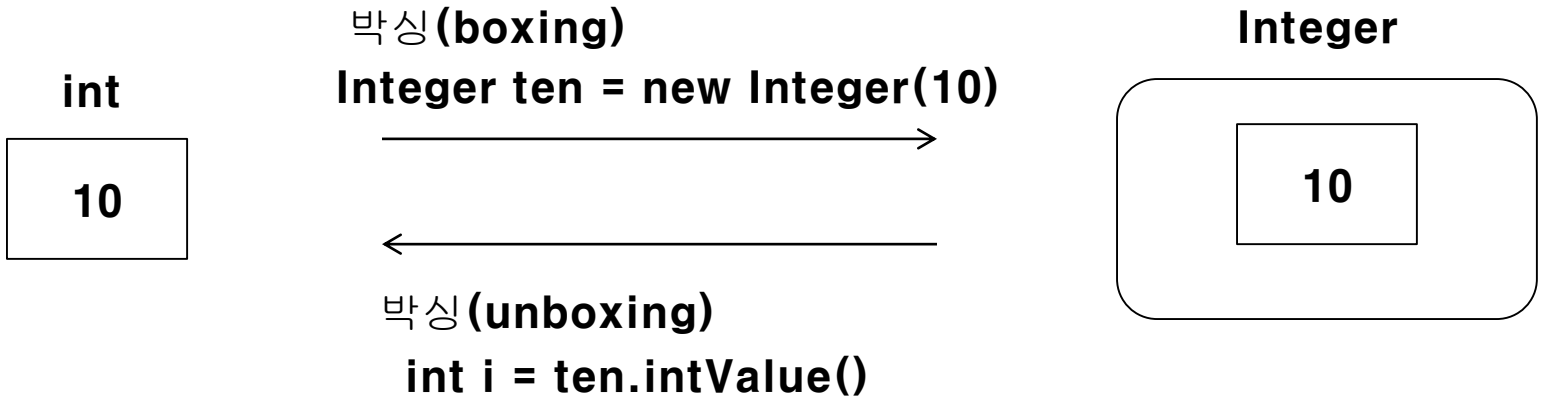
4. Wrapper 클래스 - 활용

- 타입간의 변환
- 문자열을 기본 데이터 값으로 변환 또는 그 반대(기본 데이터값을 문자열로)
- 대부분 static 함수

```
public class LangClassTest {

  public static void main( String[] args ) {
    Integer i = new Integer( 10 );
    Character c = new Character(  '4'  );
    Double d = new Double( 3.1234566 );

    System.out.println( Character.toLowerCase( 'A' )); // 대문자 A를 소문자로 변환
    if(Character.isDigit( c )){ //문자 c 가 숫자를 나타내면
       System.out.println( Character.getNumberValue( c ) ); // 문자를 숫자로 변환
    }
    System.out.println( Integer.parseInt(  "-123"  ) ); // 문자열을 정수로 변환
    System.out.println( Integer.parseInt(  "10", 16 ) ); // 16진수 문자열을 정수로 변환
    System.out.println( Integer.toBinaryString( 28 ) ); // 2진수로 표현된 문자열로 변환
    System.out.println( Integer.bitCount( 28 ) ); // 2진수에서 1의 개수
    System.out.println( Integer.toHexString(  "28"  ) ); // 16진수 문자열로 변환
    System.out.println( i.doubleValue() ); //  정수를 double로 변환
    System.out.println( d.toString() ); // Double을 문자열로 변환
    System.out.println( Double.parseDouble( "44.13e-16"  ); // 문자열을 double로 변환
  }
}
```

4. Wrapper 클래스 - 박싱(boxing) 과 언박싱 (unboxing)

- 기본 데이터 타입을 **Wrapper** 클래스로 변환하는 것을 **boxing** 이라 한다.
- 반대의 경우를 **unboxing** 이라 한다.



❑ JDK1.5 이상부터는 박싱과 언박싱이 자동으로 일어남

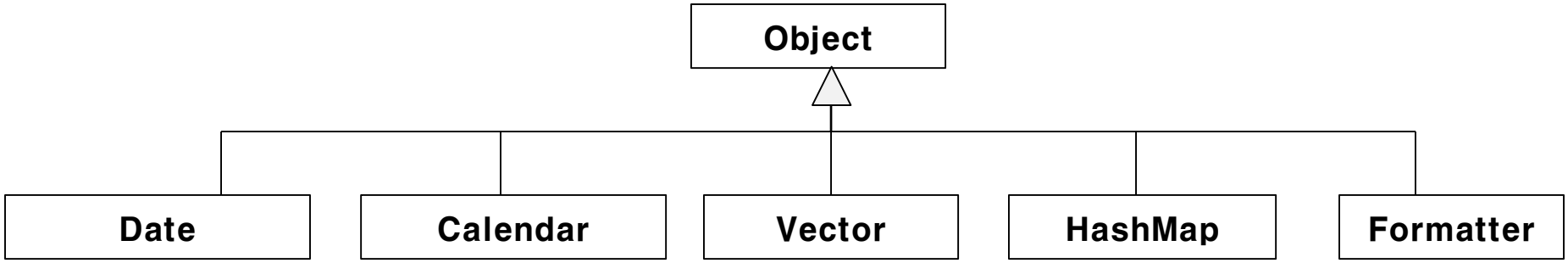
```
public class LangClassTest {

  public static void main( String[] args ) {

    Integer i = 10;
    System.out,println( i );

    int n = i + 10;
    System.out,println( n );

  }
}
```

자바기본API
2. 유틸리티 API

목차

1. 개요

- 자바 프로그램에서 필요로 하는 편리한 기능들을 모아 둔 클래스들의 패키지
- 클래스 이름 앞에 패키지 이름을 사용하지 않았다면, **Import** 를 명시적으로 해야 하는 패키지



2. Date 클래스

- 날짜와 시간에 관한 정보를 표현
- 많은 메서드들이 **deprecated(폐지예정…)** 되고 **Calendar** 클래스에서 지원

```
import java.text.DateFormat;
import java.util.Date;

public class DateTest  {
    public static void main(String[] args)  {
       Date now = new Date();
        System.out.println( now.toString()  );
        System.out.println( now.toLocaleString()  );    // Deprecated !!
     }
}

--------------------------------------------------------------------------------

DateFormat dateFormat1 = DateFormat.getDateInstance( DateFormat.FULL );
System.out.println(  dateFormat1.format( now )  );

DateFormat dateFormat2 = DateFormat.getDateInstance( DateFormat.LONG );
System.out.println(  dateFormat2.format( now )  );

DateFormat dateFormat3 = DateFormat.getDateInstance( DateFormat.MEDIUM );
System.out.println(  dateFormat3.format( now )  );

DateFormat dateFormat4 = DateFormat.getDateInstance( DateFormat.SHORT );
System.out.println(  dateFormat4.format( now )  );
```

3. Calendar 클래스

- **Date** 클래스 처럼 날짜와 시간에 관한 정보를 표현할 때 사용
- 추상 클래스이므로 직접 객체를 생성할 수 없고 **getInstance()** 메서드를 통해 **Calendar** 클래스 객체를 얻어 사용한다.
- 날짜와 시간 표현을 위해 다양한 상수를 제공하고 있다.

3. Calendar 클래스 – 활용 I

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

      Calendar caledar = Calendar.getInstance();
       // 지금!
      printDate( caledar );
      // 년도를 2013년으로 설정
      caledar.set(Calendar.YEAR, 2013);
      printDate( caledar );
      //년,월,일을 설정함(2013-6-12), 월은 0부터 시작하므로 -1을 해줘야한다.
     caledar.set(2013,6,12);
     printDate( caledar );
     //년,월,일,시,분을 설정(2013-6-12 오후 1:59)
     caledar.set(2013,6,12,13,59);
     printDate( caledar );
     //년,월,일,시,분,초를 설정(2013-6-12 오후 1:59:30)
     caledar.set(2013,6,12,13,59,30);
     printDate( caledar );

  }

  public static void printDate( Calendar caledar ) {
     System.out.println( caledar.get( Calendar.YEAR ) + "년  " +
                          ( caledar.get( Calendar.MONTH ) + 1 ) + "월 "+  //0부터 시작함
                         caledar.get( Calendar.DATE ) +"일 "  +
                         ( caledar.get( Calendar.AM_PM ) == 0 ? "오전 " : "오후 " ) + //오전:0, 오후:1
                         caledar.get( Calendar.HOUR ) + "시 "  +
                         caledar.get( Calendar.MINUTE ) + "분 " +
                         caledar.get( Calendar.SECOND ) + "초" );
     }
}
```

3. Calendar 클래스 – 활용 II

```
public class CalendarTest  {

  public static void main(String[ ] args)  {

       //오늘
      caledar = Calendar.getInstance();
      printDate( caledar );

      // 100일 후
      caledar.add(Calendar.DATE, 100);
      printDate( caledar );

     //다시 오늘
     caledar = Calendar.getInstance();
     printDate( caledar );

     // 10달전
     caledar.add(Calendar.MONTH, -10);
     printDate( caledar );

     //다시 오늘
     caledar = Calendar.getInstance();
     printDate( caledar );

     //오늘은 올해 몇일째 되는날?
     System.out.println( "오늘은  올해 " + caledar.get(Calendar.DAY_OF_YEAR) + "일째 되는 날입니다." );
  }
}
```

4. Formatter 클래스

- 데이터의 형식화된 문자열을 생성하는 클래스
- 형식문자열 **(format** 지정**)**
- 문자열 생성을 위하여 **Appendable** 인터페이스를 구현한 클래스 객체**(StringBuffer** 객체**)** 를 **Formmater** 생성자에서 등록한다**.**

```
StringBuffer sb  =  new StringBuffer();
Formatter f = new Formatter( sb );
String name = "안대혁";
int  score = 100;
f.format(  "%s님의 점수는 %d 입니다.", name, score );

--------------------------------------------------------------------------------

String s.format(  "%s님의 점수는 %d 입니다." , name, score );
```

4. Formatter 클래스 - 활용

```
import java.util.Formatter;

public class FormatterTest {

    public static void main(String[ ] args) {
       StringBuffer sb  =  new StringBuffer();
       Formatter f = new Formatter( sb );

       int i = 10, j = 20, k = i + j;
       f.format(  "%d 과  %d을 더하면  %d됩니다.₩n", i, j, k );
      System.out.print( sb );


      String s = String.format(  "%d 과  %d을 더하면  %d됩니다.₩n", i, j, k );
      System.out.print( s );

      Calendar calendar = Calendar.getInstance();
      System.out.print( String.format( "%tY년 %tB월 %te일 %b %n", calendar, calendar, calendar ) );
      System.out.print( String.format( "%tk시 %tM분 %tS초 %n", calendar, calendar, calendar ) );

      System.out.println( String.format( "실수표기법  %6.2f, %6.4f", 4.2222222, 4.222222 ) );

    }


}
```

자바기본API
3. 콜렉션프레임워크(자료구조)

목차

1. 개요

❑ 컬렉션 프레임웍(collection framework)

– 데이터 군(群)을 저장하는 클래스들을 표준화한 설계

– 다수의 데이터를 쉽게 처리할 수 있는 방법을 제공하는 클래스들로 구성

– JDK1.2부터 제공

❑ 컬렉션(collection)

- 다수의 데이터, 즉, 데이터 그룹을 의미한다.

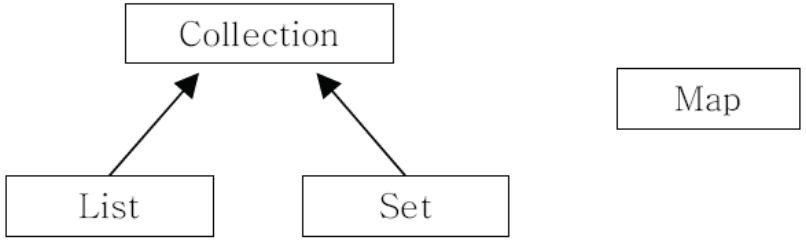
❑ 프레임웍(framework)

- 표준화, 정형화된 체계적인 프로그래밍 방식

❑ 컬렉션 클래스(collection class)

- 다수의 데이터를 저장할 수 있는 클래스(예, Vector, ArrayList, HashSet)

컬렉션 프레임웍의 핵심 인터페이스

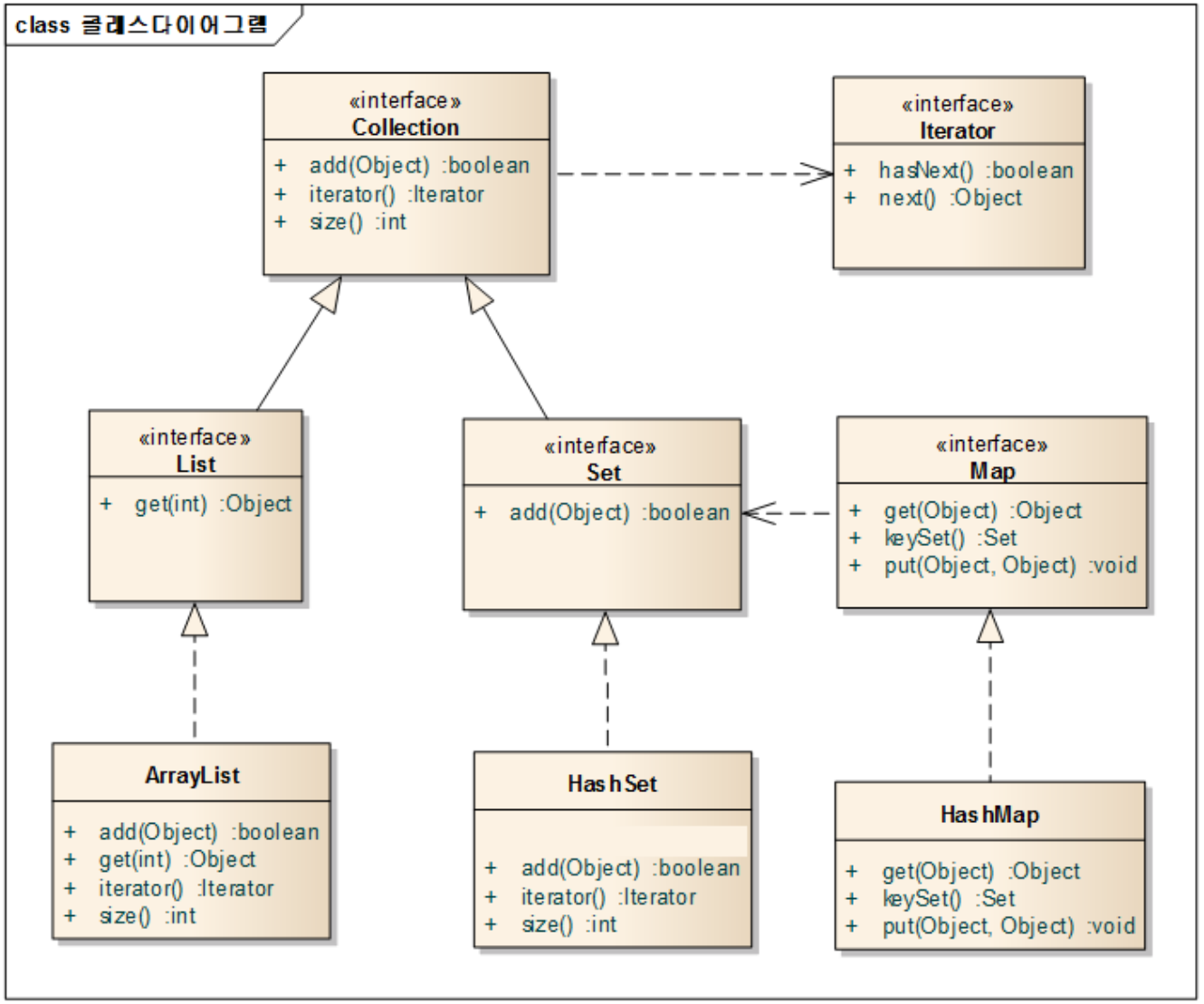


[그림11-1] 컬렉션 프레임웍의 핵심 인터페이스간의 상속계층도

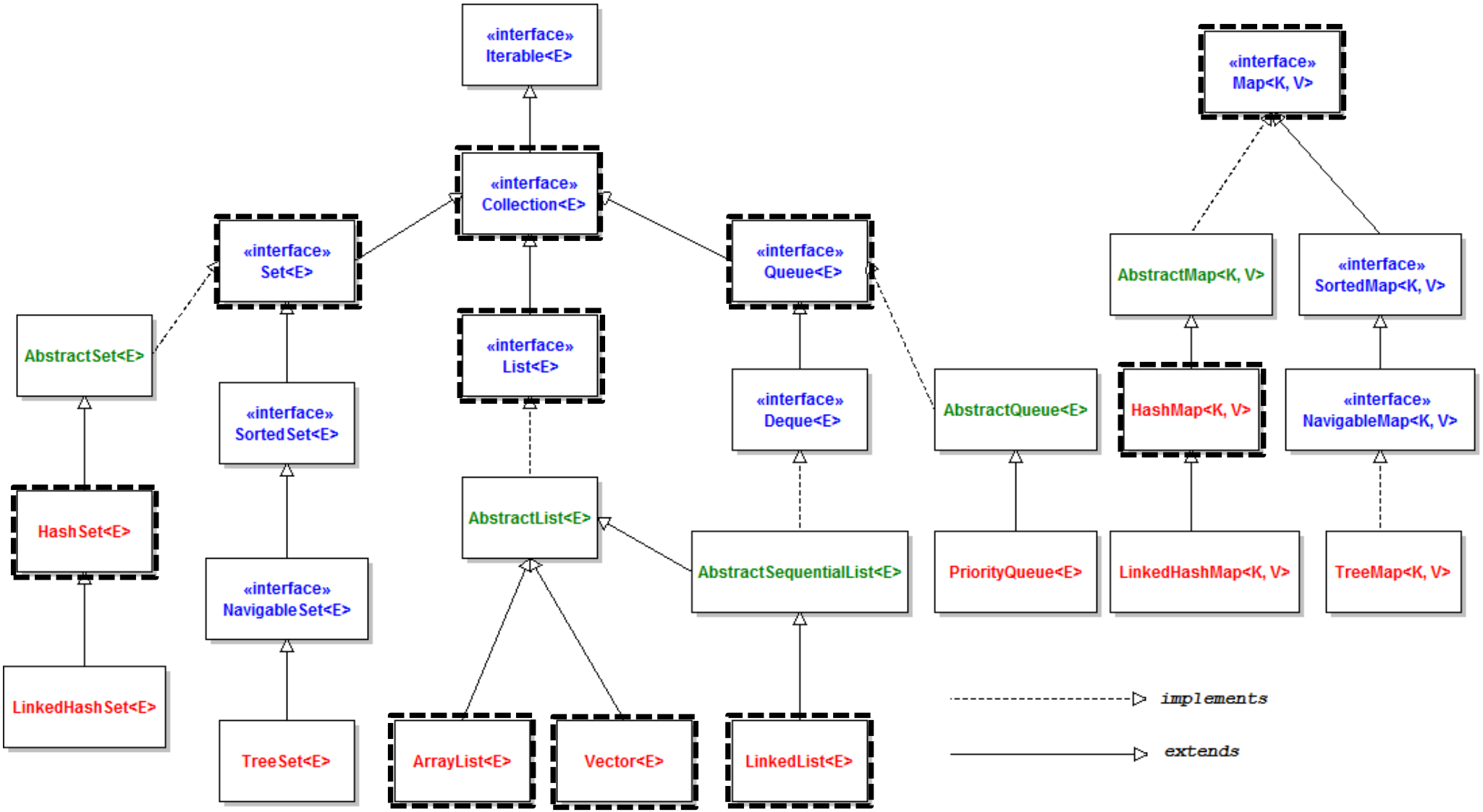
인터페이스	특 징
List	순서가 있는 데이터의 집합. 데이터의 중복을 허용한다. 예) 대기자 명단
	구현클래스 : ArrayList, LinkedList, Stack, Vector 등
Set	순서를 유지하지 않는 데이터의 집합. 데이터의 중복을 허용하지 않는다. 예) 양의 정수집합, 소수의 집합
	구현클래스 : HashSet, TreeSet 등
Map	키(key)와 값(value)의 쌍(pair)으로 이루어진 데이터의 집합 순서는 유지되지 않으며, 키는 중복을 허용하지 않고, 값은 중복을 허용한다. 예) 우편번호, 지역번호(전화번호)
	구현클래스 : HashMap, TreeMap, Hashtable, Properties 등

[표11-1] 컬렉션 프레임웍의 핵심 인터페이스와 특징

Collection Class Diagram I



Collection Class Diagram II



❑ ArrayList

- Vector, LinkedList, ArrayList 모두 추상 클래스 AbstractList를 상속받은 동적 자료구조임.
- add(int index, E element), set(int index, E element), get(int index), remove(int index) 등의 메소드를 제공하여 배열을 다루는 것과 유사한 방식으로 동적 자료구조를 사용할 수 있도록 함.
- 벡터와 리스트의 가장 큰 차이점은 자료구조에 항목을 삽입하고 삭제하는 동작이 동기화 (synchronization)되어 수행되는지 여부임.
    - 벡터는 동기화된 삽입 삭제 동작 제공
    - 리스트는 삽입과 삭제가 동기화되어 있지 않음 – 외부적인 동기화 필요

9. 이뉴머레이션(Enumeration)과 이터레이터(Iterator)

❑ Enumeration & Iterator

- 벡터와 해시테이블에서 존재하는 모든 요소에 대한 접근 방식을 제공해주는 인터페이스
- 자바 콜렉션 프레임웍으로 확장되면서 이터레이터**(Iterator)** 도입**(List, Set** 등**)**
- 이뉴머레이션 **(Enumeration)**
    - **hasMoreElements()**와 **nextElement()** 두 개의 메소드 제공
    - **Vector::elements()**
    - **Hashtable::keys()**
    - **Hashtable::values()**
- 이터레이터 **(Iterator)**
    - **hasNext(), next(), remove()** 메소드 제공
    - **Set::iterator()**
    - **List::iterator()**

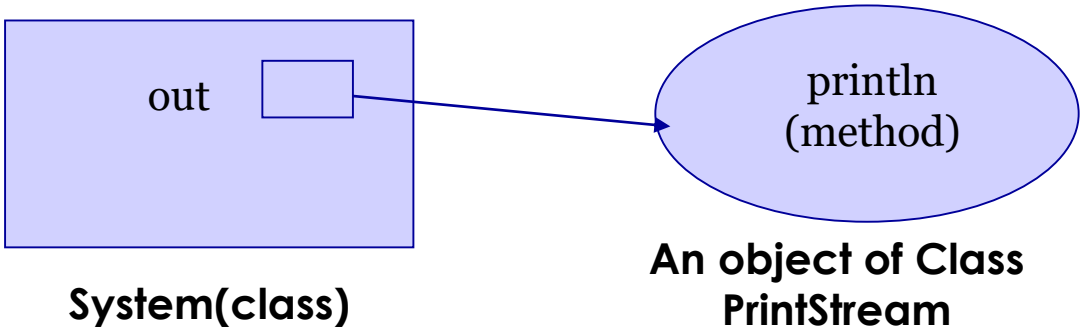
자바기본API
4. 스트림과 IO

목차

❑ I/O (Input/Output; 입출력)

- 프로그램과 외부 소스 혹은 목적지와의 데이터 및 정보의 교환
- 입력: 키보드, 파일 등으로부터 프로그램으로 들어오는 데이터
- 출력: 화면, 파일 등으로 프로그램으로부터 나가는 데이터

```
class HelloJAVA {
    public static void main (String[] args) {
      System.out.println("Hello, JAVA");
    }
}
```

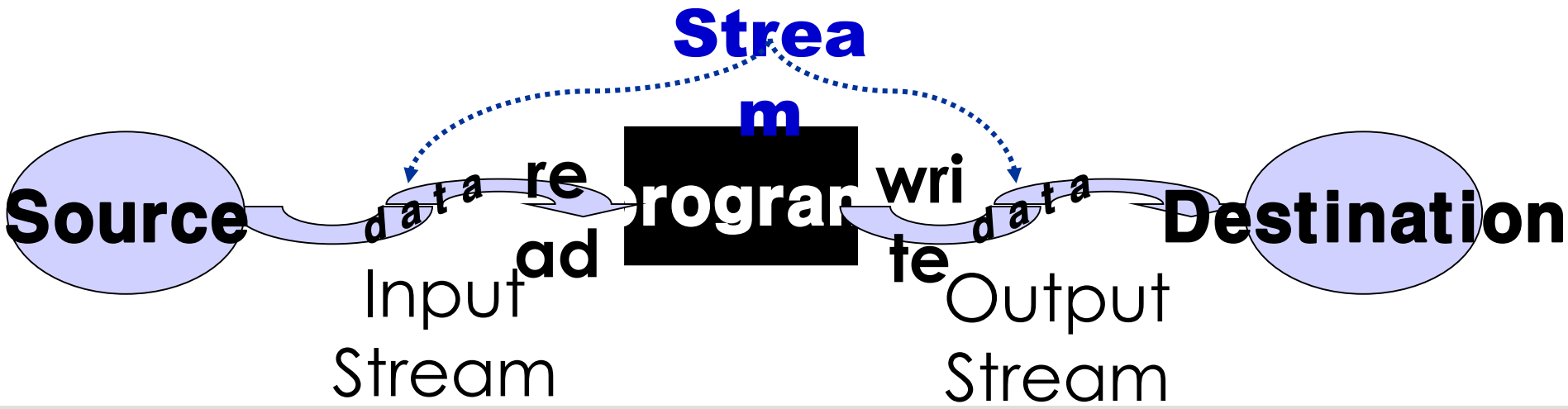


- System.out – PrintStream 객체를 참조하는 System Class의 static 변수
- System.in – InputStream 객체를 참조하는 System Class의 static 변수

1. 스트림 (Stream)과 I/O

❑ 스트림 (Stream)

- 프로그램과 I/O 객체를 연결하여 데이터를 소스에서 전달하거나 목적지로 수신하는 길(Path)
  - 목적지/소스: 키보드 및 스크린을 포함한 파일, Network connections, 다른 프로그램 등
- 한 방향, 즉 일방통행 (Unidirectional)으로 프로그램과 I/O 객체를 연결
  - Input Stream – 데이터를 읽어 들이는 객체 — *Ex)* System.in: 프로그램과 키보드 연결
  - Output Stream – 데이터를 써서 내보내는 객체 — *Ex)* System.out: 프로그램과 스크린을 연결
- Java는 Stream 또한 객체로 취급 (OOP)



1. 스트림과 I/O – 바이트 스트림(Byte Stream) vs. 문자 스트림 (Chracter Stream)

❑ 바이트 스트림(Byte stream)

– **1**과 **0**으로 구성된 **Binary** 데이터의 입출력 처리를 위한 스트림 **Ex)** 이미지, 사운드 등

❑ 문자 스트림(Character stream)

– 문자, 텍스트 형태의 데이터 입출력 처리를 위한 스트림 **Ex)** 단순 텍스트, 웹 페이지, 키보드 입력 등

– 문자 각각 **1 Byte**의 아스키코드 **(ASCII Code)** 할당

❑ 예: Number 127 (decimal)

- **Text file : 3 bytes (“1”, “2”, “7”)**로 해석하고 각각 하나의 **ASCII Code** 할당 **(decimal: 49, 50, 55, octal: 61, 62, 67)**
  1. **ASCII: 0000000 061 062 067 011 163 155 151 154 …**
  2. **Text File: 1 2 7 \t s m i l …**
- **Binary file: 1**과 **0**으로만 해석
  - **One byte (byte): 01111110**
  - **Two bytes (short): 00000000 01111110**
  - **Four bytes (int): 00000000 00000000 00000000 01111110**

❑ JAVA I/O 클래스

– **java.io** 패키지에 **I/O**를 위한 **4**개의 추상 클래스 **Reader, Writer, InpustStream, OutputStream** 정의

– 해당 클래스에서 상속받아 파일에 데이터를 읽고 씀

1. 스트림 (Stream) 과 I/O – I/O 클래스

❑ JAVA I/O 클래스

- java.io 패키지에 I/O를 위한 4개의 추상 클래스 Reader, Writer, InpustStream, OutputStream 정의
- 해당 클래스에서 상속받아 파일에 데이터를 읽고 씀

❑ Reader 클래스

- 문자 입력 스트림 (Character Input Stream)을 읽어 들이는데 사용
- read(): 하나의 문자나 문자 배열을 읽어 들임
- 추상 클래스이므로 구체적인 하위 클래스로 상속시켜서 사용

❑ BufferedReader 클래스

- a subclass of Reader
- readLine(): 한 줄 전체를 String으로 읽어 들임

❑ 키보드 입력 읽기

- System.in을 통해 Stream 으로 들어옴 →
- 입력을 문자 스트림으로 읽어 들임 →
- 한 줄 전체를 읽어 들임 →

	input	output
Byte Stream	`InputStream`	`OutputStream`
Character Stream	`Reader`	`Writer`

1. 스트림 (Stream) 과 I/O – 키보드 입력

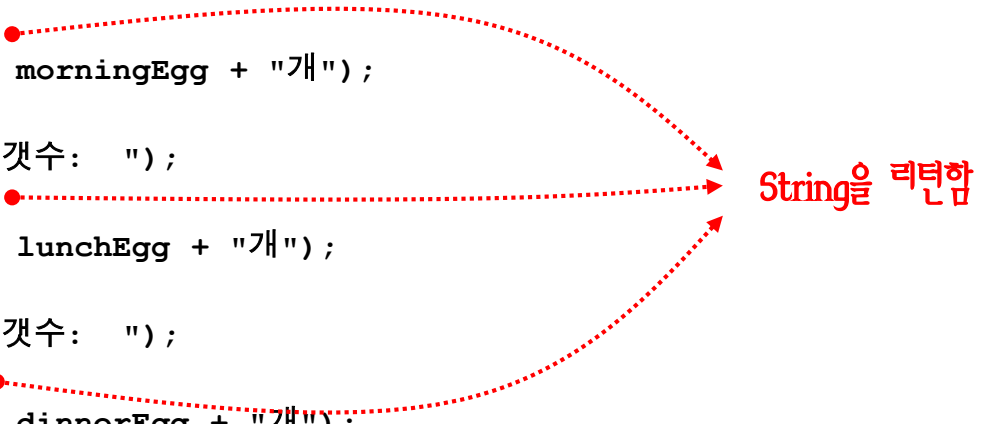
```
import java.io.*;

class readKBLine() throws IO Exception {
   public static void main(String[] arg) throws Exception {
         try {
           BufferedReader keybd = new BufferedReader(new InputStreamReader(System.in));
           System.out.println(keybd.readLine());
         } catch (IOException e) {}
   }
}
```

- BufferedReader 클래스의 생성자 (Constructor)
  - BufferReader(Reader in)
- InputStreamReader 클래스의 생성자 (Constructor)
  - InputStreamReader(InputStream in)
- InputStream
  - System.in 자체가 InpustStream 객체임

BufferRerader(Reader in) 생성자 선언 과 BufferedReader(new InputStreamReader(...)) 호출에서
Q1) InputStreamReader 객체 인자로 호출해도 BufferedReader 생성자가 동작하는 이유?
Q2) Reader 객체를 직접 사용하지 않고 굳이 InputStreamReader 객체를 사용하는 이유?

1. 스트림 (Stream) 과 I/O – 예제

```
import java.io.*;

public class eggMonster {
   public static void main(String[] args) {
     String morningEgg;
     String lunchEgg;
     String dinnerEgg;

     try {
           BufferedReader keybd = new BufferedReader(new InputStreamReader(System.in));
           System.out.print("아침에 먹은 계란 갯수:  ");
           morningEgg = keybd.readLine();
           System.out.println("아침에 계란" + morningEgg + "개");
           System.out.println();
           System.out.print("점심에 먹은 계란 갯수:  ");
           lunchEgg = keybd.readLine();
           System.out.println("점심에 계란" + lunchEgg + "개");
           System.out.println();
           System.out.print("저녁에 먹은 계란 갯수:  ");
           dinnerEgg = keybd.readLine();
           System.out.println("저녁에 계란" + dinnerEgg + "개");
     } catch (IOException e) {
           e.printStackTrace();
           System.exit(1);
     }
   }
}
```



1. 스트림 (Stream) 과 I/O – Integer 클래스

```
import java.io.*;

public class eggMonster {
   public static void main(String[] args) throws IOException { // Exception 처리를 Java에 맡김
     String morningEgg;
     String lunchEgg;
     String dinnerEgg;
      int totalEgg;

    BufferedReader keybd = new BufferedReader(new InputStreamReader(System.in));
     System.out.print("아침에 먹은 계란 갯수:  ");
     morningEgg = keybd.readLine();
     System.out.println("아침에 계란" + morningEgg + "개");
     System.out.println();
     System.out.print("점심에 먹은 계란 갯수:  ");
     lunchEgg = keybd.readLine();
     System.out.println("점심에 계란" + lunchEgg + "개");
     System.out.println();
     System.out.print("저녁에 먹은 계란 갯수:  ");
     dinnerEgg = keybd.readLine();
     System.out.println("저녁에 계란" + dinnerEgg + "개");
     System.out.println();

    totalEgg = [redacted]

    if (totalEgg > 10) { System.out.println("당신은 계란 귀신이 아니고 계란 돼지입니다!")};
  }
}
```

※ Parsing

- 보통 키보드에서 input을 읽어 들이거나 파일로 저장하는 과정에서 해당 Data를 text 형태로 처리함
- 이렇게 text로 Encoding된 Data를 원래 Data 포맷으로 Decode하는 것을 “Parsing”이라고 함

2. 파일 입출력(File I/O)

❑ 파일을 바라보는 2개의 시선 (Java vs. Operation System)

- **Java:** 스트림(Stream) 객체로 취급 → 자료형과 변수가 필요함 **Ex) PrintWriter txtStream;**
- **Operating System:** 저장된 이름 그대로 취급 **Ex) phone.txt**

❑ 텍스트 파일을 다루는 주요 클래스

	OS식 파일 처리	Java식 스트림 처리
input	`FileReader`	`BufferedReader`
output	`FileOutputStream (or FileWriter)`	`PrintWriter`

- "OS식 파일 처리"와 "Java식 스트림처리"의 연결이 필요

```
PrintWriter outTxt = new PrintWriter(new FileOutputStream("c:\\temp\\phone.txt"));
BufferedReader br = new BufferedReader(new FileReader("c:\\temp\\phone.txt"));
```

2. 파일 입출력(File I/O) - 예제

```
import java.io.*;

public class inputBufferedReader {
    public static void main(String[] args) {
      String name = null;
      String phoneNumber = null;
      String phoneList = null;

      try {
              PrintWriter outTxt = new PrintWriter(new FileOutputStream("c:\\temp\\phone.txt"));
              BufferedReader kb = new BufferedReader(new InputStreamReader(System.in));

              for (int count = 1; count <= 3; count++) {
                      System.out.print("이름: ");
                      name = kb.readLine();
                      System.out.print("전화번호: ");
                      phoneNumber = kb.readLine();
                      outTxt.println(name + " " + phoneNumber);}
              outTxt.close();
              System.out.println("... 전화번호가 c:\\temp\\phone.txt에 저장되었습니다.");

              BufferedReader br = new BufferedReader(new FileReader("c:\\temp\\phone.txt"));
              System.out.println();
              System.out.println("c:\\temp\\phone.txt의 내용은 다음과 같습니다...");
              while ((phoneList = br.readLine()) != null) {
                      System.out.println(phoneList);
              }
      } catch (FileNotFoundException e) {
              System.out.println("Error opening the file." + e.getMessage()); System.exit(0);
      } catch (IOException e) {
              e.printStackTrace();System.exit(1);        }
    }
}
```

2. 파일 입출력(File I/O) – 파일 닫아주기

- **정상적으로 수행되었을 경우, Java가 알아서 파일을 close 해줌**
- **close() 메소드를 불러주는 이유**
  - 비정상적으로 프로그램이 끝났을 경우 파일이 손상되는 것을 막아줌
  - 파일에 출력한 이후 close해주어야 읽어 들일 수 있음

    ※특수한 경우, RandomAccessFile 클래스 객체를 사용해서 파일에 출력하는 것과 동시에 읽어 들일 수 있으나, 일반적으로 Transaction 관점에서 바람직하지 않음

4. Scanner 클래스

❑ Scanner

- Java 5.0부터 java.util.Scanner 제공
- BufferedReader와 FileReader, StringTokenizer의 조합대신 사용
- 키보드 입력 사용

```
Scanner skb = new Scanner(System.in);
System.out.print("Enter name: ");
String name = stdin.nextLine();
System.out.print("Enter age:  ");
int age = stdin.nextInt();
```

- File에서 사용

```
Scanner sf = new Scanner("inputFile.txt");
```

4. Scanner 클래스 – 예제

```
import java.io.*;
import java.util.*;

public class ScannerTest {

    public static void main(String[] args) {

      Scanner sf = new Scanner("c:\\temp\\phone.txt");
      String name;
      int number1;       // 전화번호 첫번째 자리
      int number2;       // 전화번호 두번째 자리
      int number3;       // 전화번호 세번째 자리
      while (sf.hasNextLine()) {
             name = sf.next();       // 첫번째 Delimeter(이 경우는 스페이스)까지 String으로 읽음
             number1 = sf.nextInt();             // Integer값으로 Parsing
             number2 = sf.nextInt();
             number3 = sf.nextInt();
             System.out.println(name + number1 + number2 + number3);
      }
    }
}
```

4. Scanner 클래스 – Scanner vs. BufferedReader

❑ **Scanner 클래스가 있는데도, BufferedReader 클래스를 InputStream, InputStreamReader, FileReader, StringTokenizer 등과 조합해서 사용하는 이유?**

– 하나의 업무를 하나의 클래스에서 수행해서, 클래스의 변경없이 다양한 경우에 조합해서 사용 가능함

– Scanner 클래스로 처리가 가능한 경우는 Scanner 클래스를 활용할 것을 추천함

	Scanner	BufferReader
Parsing	▪ Parsing 자료형에 따라 nextInt(), nextFloat(), next() 등 사용	▪ Read(), readLine() 등의 메소드로는 Parsing 지원하지 않음 ▪ StringTokenizer와 Wrapper 클래스 메소드 사용 Ex) Integer.parseInt(Delimeteredtoken)
EOF/EOS*	▪nextLine(), nextInt()는 exception을 throw함 ▪hasNextLine(), hasNextInt() 등을 사용해서 미리 확인할 필요	▪readLine() – null 값은 return ▪read() – "-1"값을 return

***EOF/EOS – End of File / End of Stream**

감사합니다.