1. 객체지향

- 객체는 우리가 현실에서 투영할 수 있는 명사형의 모든것
- 이러한 객체를 프로그램에 반영함으로써 좀더 효율적인 코드를 얻기 위함
- 객체지향적인 프로그래밍을 통해 여러 장점을 얻을수 있다.
(1) (가독성) (2) (재사용성) (3) 무결성 (4)순차지향보다 더 짧고 읽기 쉬운 프로그래밍이 가능

2. 클래스

- 인스턴스를 생성하는 위한 틀
- (객체) = 인스턴트 - new ObjectName(), Constructor 생성자
- (속성) = 필드/멤버, 클래스 내에 멤버변수/필드로 선언할 수 있다.
- (기능) = 메서드

필드(속성) - 접근제한자 타입 변수명 [=초기값];

- 클래스 내부에서 선언되는 변수
- 변수는 기본형, 참조형 둘다 선언이 가능
- 반대로 메서드에서 선언되어지는 변수 "지역변수"
- 초기값이 올 경우는 객체가 생성되는 즉시 해당 초기값으로 필드가 초기화 된다.
- 필드가 초기화 되지 않을 경우 정수 = 0, 실수= 0.0, 논리형= false, 문자형은 0000\\u로, 참조형은 null로 초기화 된다.
- null이란 존재하지 않는 값

메서드(기능) - 접근제한자 리턴타입 메서드명(인자타입 인자변수){}

- 클래스 내부에서 실제로 동작되어지는 단위
- 메서드는 리턴타입과 인자값에 의해서 형질이 결정된다
- 리턴타입은 복수로 선언이 불가능
- 인자값은 복수로 선언이 가능하고 기본형 참조형 둘다 선언이 가능
- 객체의 기능을 정의하는 요소를 메서드라는 형태로 정의 할 수 있다.
- 리턴타입과 인자값에 의해 메서드는 다양하게 쓰일 수 있다.
- 메서드 중에서는 실제 필드의 직접적인 접근을 간접적으로 제공하기위해 쓰이는 메서드들이 있는데 이것을 getter/setter 메서드라고 한다.
- 단순히 정보를 전달하기 위해 getter/setter 메서드와 필드만으로 이루어진 클래스를 Bean 클래스라고 명명한다.
- 메서드는 같은 이름으로 인자의 타입과 갯수만 다르게 해서 정의 할 수 있는데 이러한 기법을 메서드 오버로딩이라고 한다.
- 디폴트 생성자는 인자가 존재하지 않고 내부에 기능이 존재하지 않는 생성자의 형태를 띈다.
- 생성자 또한 오버로딩이 가능하며 여러개의 생성자를 인자값만 바꿔서 선언하는 것이 가능하다.

생성자 - 접근제한자 클래스명(인자타입 인자변수)

- 인스턴스가 선언될 때 최초로 접근되어지는 메서드
- 생성자 안에서 생성자의 선언은 가능하지만 일반 메서드 안에서 생성자 선언은 불가능하다.
- 인스턴스가 선언되려면 반드시 생성자 메서드를 호출하여야 한다.
- 생성자가 선언되지 않았을 경우 JVM에서는 해당 클래스의 임의의 생성자가 있다고 판단, 디폴트 생성자를 가정하여 인스턴스를 호출한다.
- 생성자를 선언할 경우 기존의 디폴트 생성자는 소멸한다.

3. 상속 - 클래스명 extends 부모클래스명

- 한 클래스의 형질을 다른 클래스에서 그대로 물려받는 것
- 물려주는 클래스를 부모 클래스라 명명하고 물려 받는 클래스를 자식 클래스라고 한다
- 같은 기능의 재사용성을 높이기 위해서 사용
- 자식 클래스는 부모클래스의 형질을 모두 물려받지만 부모클래스에 존재하는 필드나 메서드를 전부 선언할 필요가 없다.
- 상위에 여러 부모 클래스가 있을 경우 맨 마지막 클래스는 상위의 모든 속성과 기능을 그대로 물려받는다.
- 생성자는 상속될 수 없으며 오로지 자식클래스에서 부모클래스 생성자의 참조만이 가능

Overload

- 같은 메서드명으로 인자타입과 인자 갯수만 바꿔서 여러개를 호출하는 방식
- 기능의 통일성과 가독성을 위해 사용

Override

- 부모클래스에서 상속받은 메서드를 자식클래스에서 **재선언**하는 방식
- 메서드 사용의 유연성을 위해 사용

4. 추상클래스

- **기능이 정의되지 않은 메서드를 갖고 있는 클래스**
- **상속을 통한 클래스 기능의 확장을 용이하게 하기 위해 사용**
- 추상클래스는 (인스턴스 선언)이 불가능하다.
- 다른 클래스와 마찬가지로 필드와 메서드를 가질 수 있지만 추상화된 메서드를 따로 선언할 수 있으므로 부분 추상 클래스라고 부르기도 한다.
- 추상클래스를 상속받은 자식 클래스는 추상메서드를 강제로 Overriding 해서 구현하여야 한다.
- 사용방법 :
  클래스 : 접근제한자 뒤에 abstract라고 선언
  추상메서드 : 접근제한자 뒤에 abstract 라고 선언하고 {}안의 기능 자체를 정의하지 않는다.

5. 인터페이스

- **메서드의 기능이 전혀 선언될 수 없고 오로지 명명만 가능한 완전추상클래스**
- 인터페이스는 사용자가 클래스를 작성하기 용이하게 하기 위한 **가이드라인**
- 인터페이스는 오로지 (상수 필드)와 (추상 메서드), 디폴트 메서드, 정적 상수 필드, 정적 메서드만을 가진다.
- 인터페이스는 자식클래스에서 상속받을 때 자식클래스의 선언부 뒤쪽에
  (<implements 인터페이스명>) 을 선언한다.
- 자식클래스는 인터페이스에서 선언한 추상메서드들을 전부 강제로 (Overriding)해야 한다.
- 인터페이스는 일반클래스에서 (다중 상속)이 가능하다.
- 인터페이스는 인터페이스끼리 다중 상속이 가능하며 인터페이스끼리의 상속은 인터페이스 선언부 뒤에 <extends 인터페이스명> 을 선언하여 상속한다.
- 디폴트 메서드는 하위 클래스에서 오버라이드를 한 후에 기능을 정의하지 않을 경우 치명적인 오류가 발생한다 판단될 경우 인터페이스에서 임시적으로 제공하는 메서드
- 디폴트 메서드는 접근제한자 대신 default라는 명령어를 사용

* Object
- 모든 클래스의 부모클래스
- 클래스를 선언하게 되면 상속을 하지 않아도 자동적으로 이 부모클래스의 자원을 사용이 가능하다.
- 또한 일반 클래스에서 java.lang 패키지의 모든 클래스는 import를 통한 패키지의 선언없이 사용이 가능

6.메모리

- JVM이 OS가 실행될 때 할당받는 데이터 영역
- 크게 5가지 메모리 영역이 존재한다 (Class, Stack, Heap, Native method, PC register)

- **Class 영역** : 프로그램이 실행될 때 가장 먼저 접근되어지는 메모리 영역
  (static변수, static메서드, 클래스정보, 인터페이스) 정보 등이 저장된다.
  모든 프로그램에 의해서 공유가 될수 있다.
  객체를 생성하지 않아도 이 영역의 요소들에 접근이 가능하다.

- **Call Stack 영역** : (일반 메서드)가 사용하는 영역
  지역변수, 인자값, 임시변수 등이 저장된다.
  일반적인 메서드의 로직도 이곳에서 실행된다.
  컴파일 시 크기 및 Life Cycle이 정해지는 데이터를 저장
  맨 마지막에 호출된 메서드가 가장 먼저 소멸되므로 선입후출(First in Last out) 방식을 사용한다.
  메서드의 처리가 끝나면 메모리는 자동으로 회수된다.

- **Heap 영역** : 동적으로 할당하여 사용할 수 있는 메모리
  (new를 통해 생성된 인스턴스)가 가리키는 스택의 주소를 저장한다.
  JVM에 탑재된 GC에 의해 자동으로 메모리가 관리된다.

7. Static

- 클래스 영역에 선언되어지는 메서드나 필드를 선언할 때 사용하는 명령어
- 공용으로 사용할 경우 Static을 사용할 수가 있다.
- 장점 : 인스턴스없이 접근가능, 공유가 된다.
- 단점 : 메모리의 반환이 되지 않기 때문에 시스템의 과부하를 초래할 수 있다.

8. Final

- 클래스, 필드 혹은 메서드에 제한을 주기 위한 명령어
- 클래스에 사용시 : (상속 불가)
- 필드에 사용시 : (상수화)
- 메서드에 사용시 : (오버라이드 불가)

9. 접근제한자

- 외부에서 접근 시 해당 클래스와 멤버변수 메서드에 접근을 제한할 경우 사용하는 명령어
- 공동 프로젝트 제작시에 상대방이 허가없이 수정해서 사용할 요소를 배제하기 위해 사용
- private 같은 클래스에서만 접근 가능
- default 같은 패키지와 같은 클래스 내에서만 접근 가능
- protected 같은 클래스, 같은 패키지, 자식클래스에서만 접근 가능
- public 모두 가능

10. this, super

- **this** : 자기자신의 시작 주소를 가리키는 명령어
- this는 자신의 클래스 내부에서 사용될 때 생략될 수 있다.
- 단 지역변수나 매개변수의 이름이 필드와 동일할 경우 this를 명시해서 어디의 변수인지 지정할 수 있다.

- **super** : 자기 부모의 시작 주소를 가리키는 명령어
- super는 자식 클래스 내부에서 사용될 때 생략될 수 있다.
- 단 오버라이드 된 메서드나 생성자에서 상위의 메서드 혹은 부모클래스 생성자를 호출할 경우 super를 반드시 사용해야만 한다.
- 생성자를 호출할 경우 this나 super는 생략이 불가능하며 this(), 혹은 super()란 식으로 사용이 가능하다.

11. Call by Value, Call by Reference

- 일반 기본형 변수는 변수에서 변수로 값이 이동할때 변수값 자체가 이동하므로 각각의 변수의 값은 서로 독립적으로 운영된다.

  ex) a = 1; b = a; b++, a==b(x)
- 이러한 경우를 값을 참조한다고 하여 값에 의한 참조(Call by Value)라고 한다.

- new를 통한 인스턴스 선언의 경우, 참조형 변수에서 참조형 변수로 값이 이동할 시 해당 값은 주소를 참조하게 된다.
- 다시 말해 같은 Stack의 메모리를 참조하여 해당값은 공유가 된다.
- ex) a = new a(); b = a; b.i++, b.i == a.i(o)
- 이러한 경우는 주소에 의해 참조된다고 하여 주소에 의한 참조(Call by Reference)라고 한다.

12. 다형성

- 정의 : **하나의 참조변수로 여러가지 객체를 참조할 수 있는 것**
- **조상타입의 참조변수로 자손타입의 객체를 다룰 수 있는 것을 의미**
- ex) ParentAclass a = new ChildClass();
- 부모형 타입의 자식 객체를 생성할 경우 부모타입에 선언된 메서드와 필드만이 사용가능하다.
- 자식에 선언된 메서드와 필드를 사용해야할 경우 캐스팅 연산자로 자식클래스 형을 명시하여야 한다.

13. 어노테이션

- 주석의 의미를 담고있다. 프로그래머에게 알람용도로 사용
- 버전이 올라가면서 코드 문법 에러를 체크하고, 소프트웨어 개발 툴이 빌드나 배치시 코드를 자동으로 생성할 수 있도록 정보를 제공
- 실행시 특정기능을 실행하도록 정보를 제공
- @AnnotationName 형식
- Spring Framework나 JSP에서 많이 사용된다.

14. 제네릭

- 선언시 타입을 파라미터로 선언할 수 있도록 하는 클래스
- 일반 클래스에서 다른 타입의 클래스를 참조하고자 할 경우 마치 기본형 타입을 선언하는것처럼 타입을 선언해서 참조할 때 쓰인다.

- 타입체크에 용이하고 Casting연산자를 통한 타입변환을 제거하기 위해 사용
- 사용방법 : 접근제한자 class 클래스명<T>
- 인스턴스 선언시 : 클래스명 <제네릭타입클래스> 참조변수명 = new 클래스명<제네릭타입클래스>();

--자바확장

15. 콜렉션

- java.util에 들어있는 Java의 확장기능 중 하나
- 사람들에게 편의를 제공하는 자료구조형의 집합클래스
- 콜렉션은 1.5부터 제네릭 타입을 명시하도록 유도하고 있다.
- 배열보다 쓰기 편하다
- 인터페이스는 크게 (List), (Set), (Map)으로 나뉜다
- List와 Set은 부모 인터페이스를 Collection으로 가지지만 Map은 그렇지 않다.

- (List) : 순서가 있는 연속적인 데이터 집합
  인덱스를 통한 순차적인 접근이 가능
  데이터의 중복을 허용한다.
  ArrayList, LinkedList, Stack ...etc

- (Set) : 순서를 유지하지 않는 데이터의 집합
  데이터의 중복을 허용하지 않는다.
  HashSet, TreeSet ...etc

- (Map) : 키와 값의 쌍으로 이루어져 있는 데이터의 집합
  순서는 유지되지 않음
  키는 중복허용 불가
  값은 중복 허용
  Hash, TreeMap, HashTable, ...etc

- Iterator : List 혹은 Set, Map의 데이터 직렬화를 위해 사용되는 클래스
  순차적이지 않은 Set이나 Map에 순차적으로 접근할 때 사용된다.

16. 내부클래스

- 클래스 안에 클래스를 선언하고자 할 경우 사용되는 클래스
- 선언하는 클래스 형식은 동일하며 정적클래스, 인스턴스 클래스, 지역 클래스, 익명 클래스로 나뉠 수 있다.
- 정적 클래스나 인스턴스 클래스, 지역 클래스는 특수한 상황이 아니면 잘 쓰지 않으며 익명 클래스는 상당히 많은곳에서 수시로 사용하는 스킬이다

Anonymous Class

- **참조변수의 선언없이 생성자만 선언하여 사용되는 클래스**
- **단 한번만 선언되어 사용할 경우**에 이 클래스를 많이 이용하게 된다

ex) BufferedInputStream ios = new BufferedInputStream(**new FileInputStream**)

17.예외처리

- 외부의 예외에 의해 발생하는 에러를 처리하기 위한 방법
- RuntimeException(프로그래머에 의해 발생하는 에러)를 제외한 모든 Exception의 자식 클래스로 처리가 가능하다.
- 모든 Exception관련 클래스의 최상위 부모 클래스는 (Exception) 클래스이다.
- Exception클래스들의 대부분은 java.lang에 있다.

예외처리 방식

(1) try ~ catch ~ finally

- 형식 : try{ 실행문장 }
```
catch(Exception 클래스명)
  {해당 예외가 발생 시 처리 할 문장}
 catch(Exception 클래스명)
  {해당 예외가 발생 시 처리 할 문장}
 finally{
    예외 발생 여부와 상관없이 실행되어야 하는 문장
 }
```

(2) throws 이용

- 정의 : **throw를 통해 상위 메소드로 던짐으로써 상위 메소드에서 해당 예외를 처리하게끔 하는 명령어**

- throws는 해당 예외가 발생할 메소드의 뒷부분에 선언된다.
- 형식 : **접근제한자 리턴타입 메소드명(인자값) throws Exception클래스명**
- 되도록이면 throws 메소드를 호출한 상위 메소드에 try~catch~finally가 있어야 안정적으로 돌아간다.

18. Thread

- 한 프로세스 내에서 서로 독립적인 동작을 할 경우에 사용되는 일의 처리 단위
- 보통 우리가 사용하는 모든 프로세스는 하나 이상의 Thread를 가지고 있다.
- Thread 선언하는 방법

(1) (Thread 클래스)를 상속받은 클래스를 선언하여 실행
- 형식 : 접근제한자 리턴타입 클래스명 extends Thread{} 선언
- 실행 : 클래스명 참조변수 = new 생성자
  참조변수.start();
  Thread 클래스를 받게 되면 반드시 run메서드를 오버라이드 하여 run 메서드에 원하는 기능을 구현하여야 한다.

(2) (Runnable 인터페이스)를 상속받은 클래스를 Thread 클래스에 인자로 담아서 실행
- 형식 : 접근제한자 리턴타입 클래스명 implements Runnable{}
- 실행 : Thread th = new Thread(new Runnable 클래스)
  th.start();

- Thread의 상태 변화

- new : 실제 스레드가 동작되지 않은 상태, start()전 상태
- runnable : 스레드를 실행 직후 (start 메서드를 실행한 후) 동작되기 전 대기상태
- waiting/blocked : Thread를 실행 중간에 멈추게 만드는 상태
- terminated : Thread가 모든 실행을 마치고 종료한 상태

19. 입출력
- 두개의 단말기 외부 장치 혹은 내부 대상간에 데이터를 주고받는 일을 도와주는 클래스
- 위치 : java.io
- 스트림 : 데이터를 전달하는 통로, 단방향 전송밖에 안된다.
- **바이트 기반 스트림, 문자 기반 스트림**으로 나눌 수 있다.

(바이트) 기반 스트림
- **InputStream, OutputStream**
- 1바이트 단위로 데이터를 전송하고자 하는 데이터 입출력 제공

(문자)기반 스트림
- **Reader, Writer**
- 문자 기반 스트림은 2바이트 단위로 데이터를 전송하고자 하는 데이터 입출력 제공

보조 스트림
- BufferedInputStream, BufferedOutputStream,
  InputStreamReader, OutputStreamWriter
- 스트림의 기능을 향상시키거나 새로운 기능을 추가하기 위해 사용
- 보조 스트림은 단독으로 쓰일 수 없으며 반드시 다른 IO객체를 인자로 사용해야만 한다.

파일을 대상으로 하는 스트림
- **FileInputStream, FileOutputStream, FileReader, FileWriter**

NIO
- 기존 IO는 OS커널에서 제공하는 자원을 직접 접근이 불가능 하므로 속도가 다른 언어에 비해서 많이 느렸다.
- NIO 패키지에서 CPU자원의 낭비 없이 Direct Memory Accessor를 이용해서 파일을 전송할 수 있도록 클래스를 제공
- 이 클래스들이 모인 패키지가 java.nio에 있다.

channel
- 스트림에 비해 양방향 통신이 가능한 통로
- 기존 io 처럼 stream 2개를 사용하여 양방향 통신을 구현할 필요 없이 channel 하나로 양방향 통신을 구현이 가능하다.
- 일반적인 io에서도 메서드를 통해 채널로 호출이 가능

buffer
- 데이터를 담아서 전송할 수 있는 공간
- Direct Memory Accesor에 담아서 전송이 가능하므로 cpu의 자원을 전혀 사용하지 않는다.
- buffer는 non-direct 방식과 direct방식이 존재한다.

20. JDBC

- 데이터베이스에 접근하여 SQL문을 실행하기 위한 자바 라이브러리
- JDBC는 구현 클래스가 존재하지 않는 인터페이스다.
- 각 벤더사(Oracle, MariaDB, MySql)는 JDBC인터페이스를 구현한 클래스를 만들어 제공해야만 한다.
- 주요 객체

(Connection)

- 프로그램과 데이터베이스를 연결시켜주는 역할을 하는 객체
- 아이디, 비밀번호, 제품에 따른 Connect스팩을 제공하여야 한다.

(Statement)/(PreparedStatement)

- 데이터베이스에 보낼 상태값을 저장하는 객체
- DB에 요청할 동작을 해당 객체에 담아서 전송할 수 있다.

(ResultSet)

- DB로부터 수신받은 데이터를 담는 객체
- Select 문장을 통해서 나온 결과값의 처리가 주목적이 된다.

21. 네트워크

- 단말기간의 근거리 혹은 원거리 통신을 할때 쓰이는 수단.
- 통신을 통해 파일 전송, 실시간 대화 등이 가능하다.

프로토콜

- 통신 규약이라고도 하며 컴퓨터나 원거리 통신장비에서 메세지를 주고 받는 양식과 규칙의 체계
- 프로토콜은 신호체계, 인증, 오류 감지 및 수정기능을 포함할 수 있다.

OSI 7 Layer

- 프로토콜을 기능별로 나누어서 총 7단계로 구성한 표본
- 다른 프로토콜의 가장 기본적인 모델이 된다.
- OSI 7 Layer는 7단계로 나눌수 있으며 가장 하위 단계부터 상위까지 이름이 정해져 있다.

1단계 물리계층 (Physical Layer) : 네트워크의 기본 하드웨어 전송 기술을 이룬다. 하드웨어와 접목되어 들어가는 기술이기 때문에 가장 복잡하다.

2단계 데이터 링크 계층(Data Link Layer) : P2P(Point to Point) 방식의 신뢰성있는 전송을 보장하기 위해 만든 계층으로 오류제어와 흐름제어를 하는 계층.
주소값을 물리적으로 할당받는 MAC Address가 처리되는 계층이다.

3단계 네트워크 계층(Network Layer) : 여러개의 노드를 거칠때마다 경로를 찾아주는 계층, 논리적 주소인 IP를 처리하는 계층이다.

4단계 전송 계층(Transport Layer) : 양 끝단의 사용자들이 신뢰성 있는 데이터를 받기 위해 혹은 빠른 단방향 수신을 제공하기 위한 방법을 제공하는 계층.
대표적으로 TCP와 UDP를 지원하며 사용자들이 사용할 서비스를 포트로 지정하는 계층이기도 하다.

5단계 세션 계층(Session Layer)
6단계 표현 계층(Presentation Layer)
7단계 응용 계층(Application Layer)

- InetAddress : IP주소를 다루기위한 클래스

TCP 프로그래밍

- 클라이언트 서버의 통신순서

1. 클라이언트는 Socket 서버는 ServerSocket을 준비
2. 클라이언트에서 Socket을 통해 해당 서비스포트와, IP를 작성해 해당 서비스로 접근한다.
3. 서버에서는 ServerSocket.accept()라는 메서드를 이용해 클라이언트의 요청을 대기한다.
4. 서버에서 클라이언트의 요청이 오면 해당 클라이언트에 대한 Socket을 하나 생성하여 서비스를 진행한다.
ex)Socket soc = ServerSocket.accept();
5. 클라이언트와 서버의 통신이 종료가 되면 close()메서드를 통해 해당 소켓을 닫는다.
*** 소켓은 반드시 클라이언트 하나당 하나의 소켓을 가질 수 있으며 서버는 클라이언트 마다의 소켓을 생성하여야 한다.**

----------인코딩----------
제네럴 워크스페이스 하단 텍스트파일 인코딩 아더 UTF-8
제네럴 컨텐츠타입 텍스트 하단 디폴트인코딩 텍스트,자바아카이브,자바클래스파일 UTF-8
웹 CSS,HTML,JSP 인코딩 ISO을 UTF-8
웹 JSP files - Editor - Templates- New에 네임 HTML5 JSP Page 컨텍스트 New JSP패턴에 복붙

새로만들때 넥스트누르고 만든거 찾아서 간다

생성자 this

- 생성자 this는 생성자의 이름 대신 this라는 이름을 메서드 형태로 호출한다. ex) this();
- 생성자 this는 다른 생성자에서만 호출이 되며 일반 메서드에서는 호출할 수 없다.
- 만약 오버로딩으로 선언한 각기다른 인자값을 갖고 있는 생성자를 타 생성자에서 호출할 경우 해당 인자에 해당한 값을 넣어 this메서드를 호출한다.

this

- 자신의 클래스 안에서 자신이 갖고있는 메서드나 필드를 내부에서 호출할 경우 this라는 연산자가 암묵적으로 선언된다.
- this는 클래스 자기 자신을 가리키며 일반적으로 내부의 메서드나 필드를 호출할 때 생략할 수 있다.
- 단 지역변수 혹은 인자의 이름과 필드의 이름이 동일하거나 생성자를 호출해야 할 경우에는 this는 생략이 어렵다

추상 클래스

- 클래스의 기능중, 정의가 힘든 메서드가 존재할 때 해당 메서드를 추상화 시켜 하위 클래스들이 강제로 오버라이드 하게만드는 추상화된 메서드를 갖고 있는 클래스를 추상클래스 라고 한다.
- 추상클래스는, 클래스 선언부 앞에 [abstract] 라는 명령어를 붙여서 해당클래스가 추상클래스라는 선언을 할 수 있다.
- [abstract]라는 명령어를 사용하지 않은 경우 추상메서드를 만들 수 없다.

- 추상메서드는 메서드의 리턴값 앞에 [abstract] 라는 명령어를 붙여 만들 수 있다.
- 단, 추상 메서드는 기능을 선언할 수 없다.({} 사용불가)

- 추상클래스를 상속 받은 일반 클래스는 반드시 추상 클래스에 선언된 메서드를 Override받아야 하며
- 이것은 JAVA에서 반드시 강제된다.

- 추상클래스를 상속 받았을 경우에는 반드시 추상 메서드를 선언 할 필요가 없다.
- 단, 해당 추상 클래스를 상속 받은 클래스는 반드시 부모부터 자신의 윗 부모들이 모든 추상메서드를 오버라이드 하여 구현하여야 한다.

- 추상클래스 자체로 인스턴스 선언은 할 수 없다.

JVM

Class 영역 (1순위 Static 영역)
클래스정보 , 인터페이스 , 정적필드, 메서드, 변수
main(){ new Ins()}
static은 메모리가 반환이 안됨
static
값이 변하지 않는 상수타입
레퍼런스가 단 한번 선언되는 싱글턴 타입 메서드일 경우

heap 영역
레퍼런스값을 저장 (new 인스턴스, new 배열)
메모리에 대한 주소값을 저장
Ins()
garbage collector가 지켜보다 필요없는건 버림

stack영역
메서드, 지역변수, 인자값
method1(){}

call by value
값의 수정 참조
call by reference

주소의 수정 참조

제네릭
- 제네릭은 클래스와 인터페이스, 그리고 메소드를 정의 할때 타입(type)을 파라미터로 사용할 수 있도록 하는 기능이 있다.

제네릭 이점
1. 타입 체크가 가능
2. 캐스팅 연산자의 사용을 제거

기본적인 선언방식
public class 클래스명<T>
public interface 인터페이스명<T>
public class 클래스명<K,V...>
public interface 인터페이스명<K,V...>

제네릭 메서드
- 메서드에서도 파라미터 타입과 리턴타입에 제네릭 객체를 쓸 수가 있다.
- 제네릭타입은 아래와 같이 선언 가능하다.
  public <타입 파라미터> 리턴타입 메소드명(매개변수...){}
- 제네릭 메서드는 두가지 방식으로 호출이 가능하다.
1. 타입 파라미터를 명시적으로 지정하는 경우
2. 매개값을 보고 구체적 타입을 추정

throws :
- 예외를 해당 메서드에서 처리하는 것이 아닌 상위 메서드에서 처리하도록 예외를 던질 경우 메서드 선언부 뒤에 [throws 해당예외클래스]를 놓는데 이것이 throw를 통한 예외처리 방식이다.
- throws를 처리하지 않았을 경우 암묵적으로 throws Exception 이 뒤에 붙어서 처리가 되므로 상위 메서드에서 해당 에러를 받아 처리하게 된다.
- throws 뒤에 붙는 예외 클래스들은 하나 이상이 올 수가 있다. (즉 다수 예외 처리 가능)

Exception 강제 발생
- Exception은 throw라는 명령어를 통해 예외를 강제 발생 시킬수가 있다.
- 해당 예외는 반드시 Exception클래스를 상속받는 클래스가 선언되어 있어야만 한다.
- throw를 통해 해당 인스턴스 변수를 예외부분 try-catch로 던지게 되면 catch문구에서 해당 예외를 찾아 처리한다.
- Exception 클래스를 임의의 클래스에서 상속받아 커스텀화 된 예외클래스를 선언할 수 있으며, 해당 클래스 안에 로직을 넣음으로써 객체간의 기능의 분리를 시킬수 있다.

Thread
- 같은 프로세스 내에서 독립적으로 동작하는 일의 단위
- 모든 프로세스는 하나 이상의 스레드를 가진다
- 스레드의 구현 방식
  1. Thread 클래스를 상속받아서 구현하는 방식
     - Thread를 통해 구현하는 방식은 객체 선언 후에 start()메서드를 호출함으로써 run메서드에 구현된 로직을 독립적으로 실행시킬 수 있다.
  2. Runnable 인터페이스를 상속받아서 구현하는 방식
     - Runnable을 통해 구현하는 방식은 객체를 Thread 클래스 생성자의 인자로 넣어서 선언하는 방식으로 Runnable 독립적으로 스레드를 호출할 수 없으며 반드시 Thread클래스에 받아서 선언하여야 한다.
     - 스레드에는 우선순위를 결정하는 setPriority라는 메서드가 존재하며 각각 1~10 사이의 숫자로 우선순위를 지정할 수 있다.
     - 10 = MAX_PRIORITY -> 최대 우선순위
       1 = MIN_PRIORITY -> 최소우선순위