Day01_변수

1) 변수 : 프로그램 처리 과정에서 데이터를 담아 둘 수 있는 메모리의 공간 의미

int a; 데이터타입 변수명
int a = 1; 대입연산자 초기값

· 데이터 타입 : 변수에 기억시킬 데이터
· 변수명 : 기억장소 주소를 대신하여 사용할 이름
· 초기 값 : 변수를 선언한 후 기억시킬 값

2) 데이터 타입

데이터	자료형
정수	byte, short, int, long
실수	float, double
문자	char
참과 거짓	boolean

Day02_연산자

1) 연산자 : 일정한 규칙을 가지고 계산하는 것으로 우리가 일상생활에서 사용하고 있는 덧셈, 뺄셈, 곱셈, 나눗셈을 비롯하여 프로그램에서는 자동증감, 동등비교, 비트연산 등 많은 연산자가 있다.

2) 우선순위에 따른 연산자 종류

순위	명칭		연산자	결합성
①	1 차 연 산 자		( )	좌결합성 →
②	단 항 연 산 자		! ++ --	우결합성 ←
③	이항연산자	승법연산자	* / %	→
④		가법연산자	+ -	
⑤		관계(비교)연산자	< <= > >= == !=	
⑥		비트곱연산자	&	
⑦		비트합연산자	|	
⑧		논리곱연산자	&&	
⑨		논리합연산자	||	
⑩	조건(3항)연산자		? :	
⑪	할 당 연 산 자		= += -= *= /= %=	←

Day03_제어문

문(statement)	실행문	순차문	
		제어문	조건문(if, switch)
			반복문 (for, while,do~while)
	비실행문	주석(// /* */)	

1) if문

```
if(조건식1) {
        조건식1이 참일 때 실행할 명령문;
    }
    if(조건식1) {
        조건식1이 참일 때 실행할 명령문;
    }else{
        조건식1이 거짓일 때 실행할 명령문
    }
    if(조건식1) {
        조건식1이 참일 때 실행할 명령문;
    }else if(조건식2){
        조건식1이 거짓이고 조건식2가 참일 때 실행할 명령문;
    }else{
        위의 조건식 둘 다 참이 아닐 때 실행할 명령문;
    }
```

2) swith문

```
switch(정수형 또는 문자형 변수) {
        case 변수값1 : 위의 변수가 변수값1일 때 처리할 명령문; break;
        case 변수값2 : 위의 변수가 변수값1일 때 처리할 명령문; break;
        • • • • • •
        default : 위의 변수값들이 아닐 때 처리할 명령문; break;
        }
```

3) 반복문 : 반복적으로 수행하도록 하는 동작

3-1) for문 : 미리 설정된 횟수만큼 반복적으로 수행.

```
for( ①초기값 ; ②반복할조건 ; ④증감식){
    ③ 반복할 명령문들;
    }
```

3-1) while문 : for문과 동일하게 반복문의 일종이고, 조건이 만족될 때까지 반복적으로 수행

```
while(조건식){
        조건식이 참일 때 계속 실행할 명령문들;
    }
```

3-2) do-while문 : 우선 수행한 후 조건이 만족되면 수행 아니면 수행하지 않는다.

```
do{
최초 한번은 무조건 실행. 두번째부터는 조건식이 참이면 계속 수행할 명령문들;
} while(조건식);
```

Day04_배열

1) 배열 : 동일 자료형의 집합. 하나의 이름으로 다수 개의 데이터를 사용할 수 있음.

2) 배열의 선언와 생성
- 변수 선언과 거의 비슷하며, 여러 개의 데이터가 모여 있어 '{ }'를 이용 한다.
- 배열의 크기는 **최초에 한번 설정되면 변경이 불가** 하다.
- 배열을 객체로 취급.
- 배열선언 → 배열의 메모리 할당(배열 생성) → 배열이용

3) 배열의 메모리 구조
- int i=10; : 메모리에 I의 공간이 생성, i 공간 안에 10이라는 데이터가 들어있다.
- int[] iArr = {10,20,30}; : 메모리에 iArr 공간이 생성
  : iArr 공간 안에 배열을 구성하고 있는 데이터의 주소값이 들어있다.

Day05~07_객체지향프로그램

1) 객체지향 프로그램 : 프로그램(실제세계)를 객체(사물)라는 기본 단위로 나누고 이 객체들 간의 상호작용을 기본개념으로 하는 프로그램

2) 객체지향과 절차지향의 비교

비교	절자치향 언어	객체지향 언어
장점	처리속도가 빠름 컴퓨터의 처리구조와 비슷하여 실행속도가 빠름	코드의 재사용성 용이 개발이 간단함 유지보수가 쉬움 대규모 프로젝트에 적합
단점	유지보수의 어려움 대규모 프로젝트에 부적합 프로그램 분석이 어려움	처리속도가 느림 객체에 따른 용량 증가 설계단계에서 시간이 많이 소요

3) 메소드 : 작업을 수행하기 위한 명령문의 집합

4) 메소드의 장점과 작성방법 : 반복적으로 수행되는 여러 문장을 메소드로 작성한다.

```
접근제한자 [static] 리턴type 메소드명([매개변수1, 매개변수2,..]){
   //return이 없을 경우 리턴type은 void
   처리할 프로세스들
   [return 리턴값;]
}
```

5) 메소드의 한계 : 한 문서 내에 메소드의 수가 많아질 경우 추후 유지 보수에 많은 어려움이 발생

6) 객체의 개념 및 클래스의 이해
- 객체 : 같은 종류의 데이터와 메소드가 함께 있는 구성체
- 예) 객체-자동차 / 데이터(속성)-색상,배기량,속도 등 / 메소드-드라이브,주차,레이싱 등
- 프로그램에서의 객체 = 데이터 + 메소드

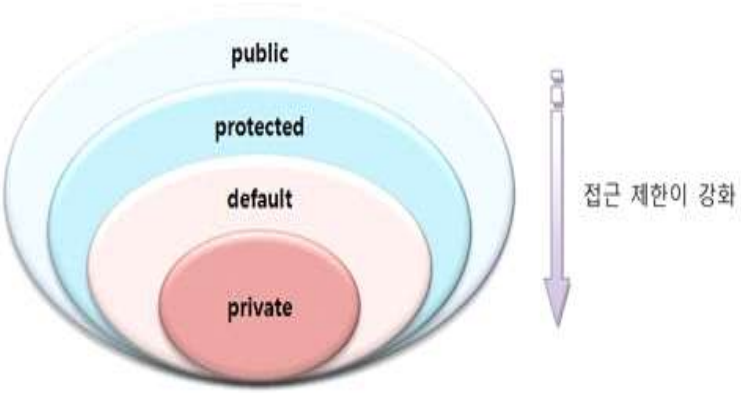
7) 클래스의 기초적인 코딩방법

step1 : 패키지명 생성하기
step2 : 클래스명 생성하기
step3 : 데이터(인스턴스 변수=멤버변수, 필드) 생성하기
// 이 데이터는 생성자나 setter를 이용해서 초기화하지 않으면 객체는 null, 숫자는 0, boolean은 false로 초기화되어 들어간다
step4: 생성자함수 생성하기
// 클래스명과 똑같이 리턴타입이 없는 메소드를 생성자라 하며
처음 클래스형 객체를 만들때 호출된다.
모든 클래스는 반드시 하나 이상의 생성자가 있어야 한다.
만약 하나도 없으면 JVM이 디폴트 생성자를 만들어 준다.
step5 : 메소드 생성하기
step6 : Getter & setter 생성하기

8) 생성자 : 매개변수 있는 생성자와 매개변수 없는 생성자 등 여러 종류의 생성자를 가질 수 있다. 성자가 없을 때는 디폴트 생성자가 컴파일러 단계에서 자동 생성한다. 한 개 이상의 생성자가 있으면 디폴트 생성자는 자동 생성되지 않는다.

Day08_접근제한과 static

1) 접근제한 : 클래스의 데이터나 메소드에 대해서 다른 클래스로부터 접근을 제한하는 것



접근 제한	적용할 내용	접근할 수 없는 클래스
public	클래스, 필드, 생성자, 메소드	없음
protected	필드, 생성자, 메소드	자식 클래스가 아닌 다른 패키지에 소속된 클래스
default	클래스, 필드, 생성자, 메소드	다른 패키지에 소속된 클래스
private	필드, 생성자, 메소드	모든 외부 클래스

2) static : 클래식변수(=static 변수)

- 클래스로부터 생성된 객체들의 수와 상관없이 하나만 생성
- 한 클래스로부터 생성된 모든 객체들은 클래스 변수를 공유
- 객체들 사이의 통신에 사용하거나 객체들의 공통속성을 나타낼 수 있음.
- 클래스 이름을 통하여 접근.
- 객체들끼리 공유하고 싶은 정보가 있을 때 사용.

Day09_패키지

1) 패키지(package) : 프로그래밍에서 여러 클래스를 관리하기 위해 기능적으로 영향을 미칠 수 있는 클래스끼리 묶어 놓고, 접근 범위 안에 효과적으로 호출하기 위하여 사용함

2) 패키지 특성 : **관련 있는 클래스들을 패키지로 그룹 지어 관리하는 것(클래스를 모아 놓은 폴더)이라는 것**을 확인할 수 있음

3) import의 이해 : 패키지를 만들고 클래스를 만든 후 서로 다른 패키지의 클래스를 사용하려고 할 때는 반드시 import 키워드를 사용해서 해당 클래스를 import, 동일한 패키지의 클래스는 import가 필요 없음

Day10~11_상속

1) 상속(Inheritance) : 부모가 가지고 있는 클래스의 속성과 메서드를 활용할 수 있는 개념으로 시작.

- 기존의 클래스를 재사용해서 새로운 클래스를 작성
- 두 클래스를 부모와 자식(조상과 자손)으로 관계를 맺어줄 수 있음
- 자손은 조상의 멤버를 상속받으나, private멤버는 직접 제어할 없다.
- 자손의 멤버 개수는 조상보다 작을 수 없다(같거나 많다)
- JAVA에서는 다중 상속이 지원되지 않는다.

2) 상속 문법의 이해

```
접근제어자 [final/abstract] class 클래스 이름 extends 상위클래스(super class) {

        추가할 멤버 변수 선언;

        생성자;

        추가할 메소드 선언;

    }
```

3) Object클래스 : 모든 클래스의 최고조상

- 조상이 없는 클래스는 자동적으로 Object클래스를 상속받음
- 상속계층도의 최상위에는 Object클래스가 위치
- 모든 클래스는 Object클래스에 정의된 11개의 메서드를 상속받음

4) 생성자 오버로딩(overloading=polymorphism 중복정의) : 인자의 타입이 다르면 같은 이름의 메소드라도 다른 기능으로 중복정의가 가능한 것

5) 오버라이드(override=재정의) : 부모 클래스의 메소드를 자식 클래스에서 재정의.
- 부모 클래스의 메소드를 자식클래스에서 동일한 이름으로 다시 재정의
> 부모클래의 메소드를 찾지 않고 자식 클래스의 메소드 호출

Day12_추상클래스

1) 추상클래스 : 강제로 부모클래스에서 자식클래스에게 메소드를 강제로 재정의(override)하도록 만들어진 클래스

2) 추상클래스의 문법

abstract(추상클래스 및 추상 메소드를 선언하는 예약어)이용

```
public abstract class ClassName {
    ...
}
```

3) 추상클래스의 특징
- 하나 이상의 추상 메소드가 포함. 추상메소드는 정의만 하고 구현은 하지 않음.
- 메소드 선언만 하고 실제로 구현은 상속받는 클래스에서 진행
- 기능은 자식 클래스에게 위임하는 것
- 추상클래스에서 정의된 추상적인 기능은 하위 클래스에서 상세 구현
- 클래스의 프레임만 구성. 직접 객체 생성 불가능(abstract는 인스턴스화를 금지하는 키워드)

4) 추상클래스가 필요한 이유 : 강제성을 느낄 때 사용

Day13~14_인터페이스

1) 인터페이스의 정의

1-1) 작업명세서
- 앞으로 만들 것을 표현해 놓은 것이며 실제 구현된 것이 전혀 없는 기본 설계도
- 일종의 추상클래스. 추상클래스(미완성 설계도)보다 추상화 정도가 높음
- 인스턴스를 생성할 수 없고, 클래스 작성에 도움을 줄 목적으로 사용
- 추상메서드와 상수만을 멤버로 가질 수 있음

1-2) 다향성을 가능하게 만듬 : 하나의 객체를 다향하게 많은 type으로 만들 수 있음

1-3) 객체를 부속품화 : 다양한 객체를 제품의 부속품처럼 개발자 마음대로 변경이 가능

1-4) 객체와 객체 간의 소통 수단

2) 인터페이스의 문법

```
public interface 인터페이스이름 {
        public static final 타입 상수이름 = 값;
        public abstract 메서드 이름(매개변수 목록);
        //구현된 메소드는 갖을 수 없다
}
```

- 모든 멤버변수는 public static final 이어야하며, 생략가능
- 모든 메서드는 public abstract 이어야 하며, 생략가능
- private는 불가 : 상수나 메소드를 만들 때 private 접근 제한자는 불가
- 구현은 Implement 되는 클래스에서 진행

3) 다향성 : 하나의 인터페이스를 사용하여 다양한 구현 방법을 제공, 하나의 클래스나 함수가 다양하게 동작하는 것

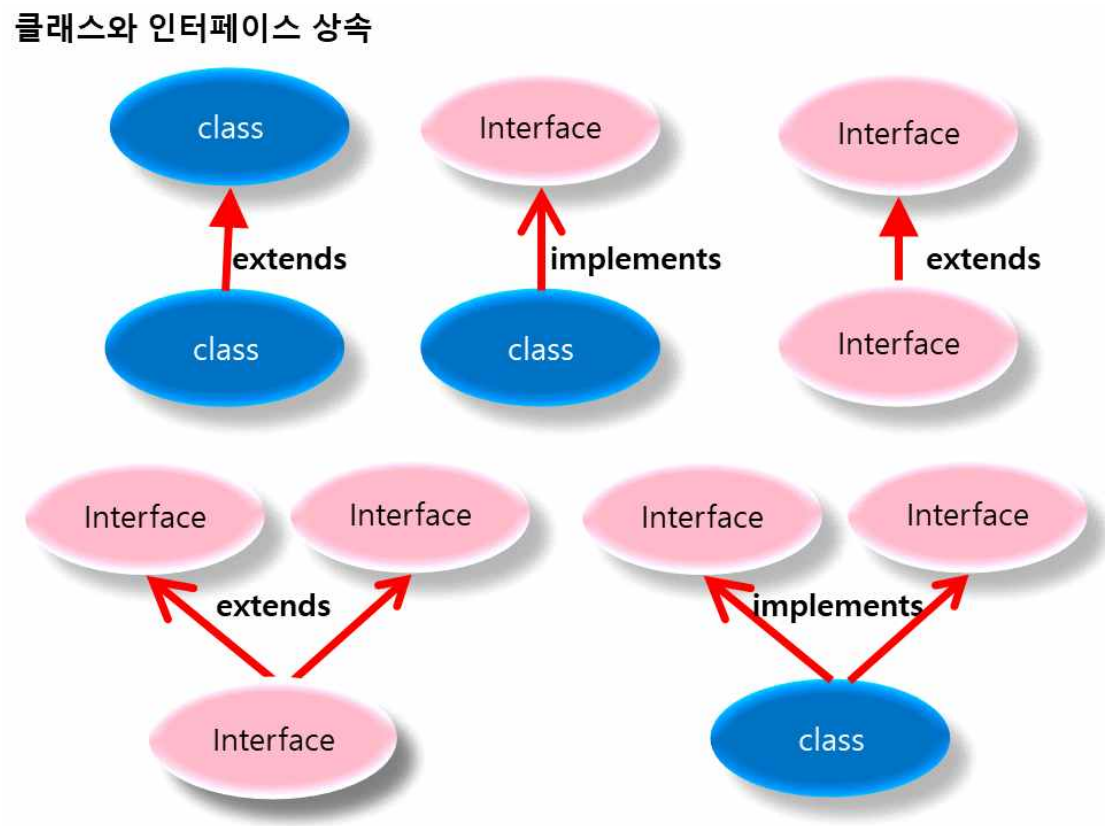
3-1) 오버로딩(overloading)
- 컴파일러 입장에서는 기존에 없는 새로운 메서드를 정의하는 것
- 메소드 다중정의 (같은 class에서 동일한 메소드가 매개변수를 달리 여러 개 존재)

3-2) 오버라이딩(overriding)
- 상속받은 메서드의 내용을 변경하는 것
- 메소드 재정의 : 부모클래스와 자식클래스에 동일한 method 존재(틀만 가져와 재정의)
오버라이딩의 조건 : 선언부가 같아야 한다(이름, 매개변수, 리턴타입), 접근제어자를 좁은 범위로 변경할 수 없다.

4) 인터페이스와 추상클래스

공통점	• 추상메소드를 가지고 있음 - 추상메소드를 가지고 있어 하위 클래스에서 구현 • 변수 타입이 목적 - 객체생성이 목적이 아닌 변수 타입을 정의하는 것이 목적
차이점	• 상속, 구현 - 추상메소드는 상속을 통한 사용이고, 인터페이스는 구현을 통한 사용 • 구성요소 차이 - 추상클래스는 일반 클래스와 동일하게 변수, 메소드의 모든 기능을 사용 가능, 인터페이스는 상수와 추상메소드만이 존재 • 단일상속, 다중구현 - 추상클래스는 상속이므로 단일 상속만 지원, 인터페이스는 다중구현이 가능

4) 클래스와 인터페이스 상속 관계



Day15_패턴을 통한 객체지향 언어의 이해

1) 패턴(디자인 패턴) : 소프트웨어 개발 시 자주 나타나는 구조나 방식, 패턴을 구체적이고 체계적으로 나누어 정리한 것

2) 패턴을 사용하는 목적

- 코드의 응집성 ↑ : 어떤 로직 부분을 변경시키고 싶을 때, 코드가 산재되어 있어서 그 부분들을 일일이 변화시켜야 하는 것이 아니라 바로 그 한부분만을 변경&디버깅 가능

- 코드의 결합성 ↓ : 코드 한 부분을 변화시키면, 다른 부분까지 변화X
  각각 맡은 로직이 깔끔하게 분리되어 있어서, 서로간에 쉽게 변경이 가능하고 대체 가능

- 코드의 재활용성 ↑ : 한번 작성한 로직을 다시 작성하지 않아도 되도록 미리 정해놓은 패턴에 따라 구성

- 코드의 효율성 ↑ : 이미 다듬어진 패턴을 이해하고 사용함으로써, 효율적인 코드를 생산

3) 싱글톤 패턴(Singleton pattern) : 어떤 클래스의 객체는 오직 하나인 유일한 객체를 만들어 여러 가지 상황에서 동일한 객체에 접근하기 위해 만들어진 패턴

4) 스트레티지 패턴(Strategy pattern) : 기능 하나를 정의하고 각각을 캡슐화하여 교환해서 사용할 수 있도록 만듬. 해당 패턴을 활용하면 기능(알고리즘)을 사용하는 클라이언트와는 독립적으로 기능(알고리즘)을 변경할 수 있음.

Day16~18_API

1) API : 자바 시스템을 제어하기 위해서 자바에서 제공하는 명령어들을 의미
Java SE(JDK)를 설치하면 자바 시스템을 제어하기 위한 API를 제공
자바 개발자들은 자바에서 제공한 API를 이용해서 자바 애플리케이션을 만들게 됨.
패키지 java.lang.*의 클래스들도 자바에서 제공하는 API 중의 하나라고 할 수 있음

2) String의 주요 기능들(메소드)

- String concat(String str) : 저장된 문자열과 str문자열을 결합
- String substring(int begin) : 시작위치부터 마지막까지의 문자열을 반환
- int length() : 문자열 길이를 반환
- String toUpperCase() : 대문자로 반환
- String toLowerCase() : 소문자로 반환
- char charAt(int index) : index 위치의 문자를 반환
- int indexOf(char ch) : 첫번째 ch문자의 위치를 반환
- int lastIndexOf(char ch) : 마지막 ch문자의 위치를 반환
- boolean equals(String str) : 지정된문자열과 str문자열이 같은지 비교
- boolean equalslgnoreCase(String str) : (대소문자구분없이)지정된문자열과 str문자열이 같은지 비교
- String trim() : 문자열 앞뒤 공백제거
- String replace(char old, char new) : 문자열 내의 old문자를 new문자로 반환
- String repalceAll(String old, String new) : 문자열 내의 old문자열을 new로 반환

3) String의 단점 : 메모리를 과소비를 하여 속도가 느려짐

4) StringBuffer & StringBuilder : 모두 객체 내부에 있는 버퍼(buffer, 데이터를 임시로 저장하는 메모리)에 문자열의 내용을 저장해 두고 그 안에서 추가, 수정, 삭제 작업을 진행. 새로운 객체를 만들지 않고도 문자열 조작을 할 수 있어 속도적인 측면에서 개선됨.

5) StringBuilder의 주요 기능들(메소드)

- append(String str) : 문자열 str 추가
- insert(int index, String str) : 특정 index에 문자열 str 추가
- delete(int start, int end) : index위치 start부터 end앞 까지 삭제
- deleteCharAt(int index) : index위치의 특정 문자 하나 삭제
- int capacity() : 문자열 크기 반환

▶ ensureCapacity(int size) : 버퍼의 크기를 size만큼 늘리는 메소드
▶ trimToSize() : 과도한 버퍼 크기를 적당하게 줄이는 메소드

6) System.currentTimeMillis() : 1970년도부터 현재까지의 밀리세컨(1/1,000초) 단위로 표시. 거의 속도 테스트 용도로 사용

7) StringTokenizer 주요 기능(메소드) : 문자열 분할

8) 날짜(Calendar와 GregorianCalendar) API
- 날짜와 시간을 표현할 때 많이 쓰이는 Calendar클래스 : 싱글톤클래스
- 날짜와 시간을 표현할 때 많이 쓰이는 GregorianCalendar클래스 : 일반클래스

9) Scanner : 키보드에서 타이핑하는 문자열 또는 입출력 프로그래밍에서 값을 읽어올 때, 무엇인가를 얻어 올 때 사용

10) Wrapper 클래스 : 기초데이터를 객체데이터로 변화시키는 클래스

11) 기초데이터에 상응하는 객체 데이터 클래스

기초 데이터	객체 데이터
byte	Byte
short	Short
int	Integer
long	Long
float	Float
boolean	Boolean
char	Char

12) Timer, TimerTask클래스
Timer객체 : 일정한 시간이 되면, TimerTask객체가 작동되도록 하거나 일정시간마다 TimerTask객체가 작동되도록 진행
TimerTask클래스 : 추상클래스이므로, TimerTask클래스를 상속받는 클래스를 만들어서 사용

Day17 예외처리

1) 예외처리의 필요성 : 필요성은 어느 한 부분에서 예외가 발생하더라도 계속해서 프로그램이 동작되도록 하는데 목적

3) 예외처리 문법(try ~catch) : 예외가 발생했을 때 자체적으로 catch문을 이용해서 해결

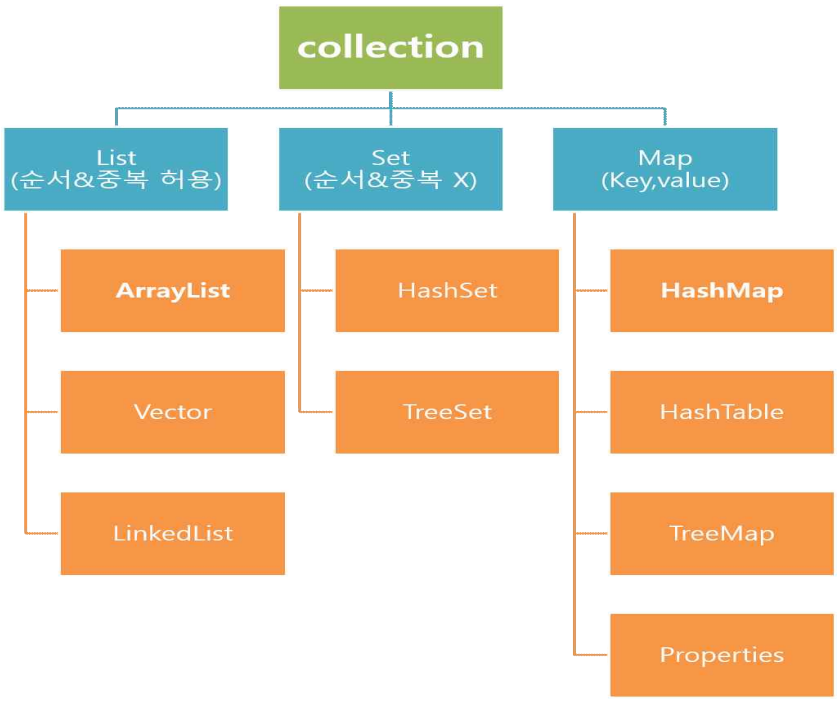
```
try {
try블럭 ; 익셉션이 발생할 가능성이 있는 명령문들(문제가 발생할 수 있는 로직을 기술)

        }catch(익셉션타입 익셉션변수) {
그 익셉션을 처리하는 명령문(try블록안에서 문제가 발생했을 때 대처방안 기술);

        } finally {
            익셉션 발생 여부와 상관없이 맨 마지막에 실행할 명령문;
        }
```

4) 예외처리 문법(throws) : 나를 호출한 쪽으로 예외를 던저버리는 방식

Day18 Java Collection

1) collection : 다수의 데이터, 데이터그룹을 의미함

2) 자바에서 제공되는 Collection 자료 구조들



3) List 계열 Collection 클래스 : 자료구조 중 가장 많이 사용하고 쉽게 사용
배열과 비슷하지만, List는 처음 만들 때 크기를 고정하지 않아도 되고 그 크기는 유동적

- ArrayList : 배열과 매우 비슷, 인덱스가 존재하며 데이터는 중복을 허용, 인덱스가 가장 중요
- LinkedList : 불연속적을 존재하는 데이터를 연결, 순차적으로 데이터를 추가/삭제할

경우 ArrayList가 빠르고 비순차적으로 데이터를 추가/삭제하는 경우 LinkedList가 빠름.
- Vector : ArrayList와 비슷하지만 속도면에서 떨어짐. ArrayList보다 멀티스레드 환경에서 안전하여 많이 사용

4) Map 계열 Collection : 인덱스 대신 키 값으로 데이터를 엑세스를 진행함. List계열과 달리 인덱스가 없고 키와 값만 있어 키값이 유니크 값.

5) Set 계열 Collection : 자료구조에서는 데이터의 순서가 없음. 중복된 데이터는 허락하지 않음. 중복된 데이터의 의미는 hashcode()값이 같거나 equal()메소드의 결과값에 의해 해석

6) Iterator(반복자) 자료구조 : 데이터를 반복적으로 검색하는데 유용한 인터페이스. 모든 자료구조형은 iterator() 메소드를 지원하고 있음.

Collection	특 징
List	순서가 있는 데이터의 집합, 데이터의 중복을 허용한다 ex. 대기자명단
	구현클래스 : ArrayList, LinkedList, Stack, Vector 등
Set	순서를 유지하지 않는 데이터의 집합. 데이터의 중복을 허용하지 않는다.
	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
	구현클래스 : HashMap, HashTable 등

Day19 자바 입출력(i/o)

1) I/O : 입력(Input)과 출력(Output)을 뜻함.
- Input : 컴퓨터에게 입력하는 것, 파일 데이터를 읽음, 키보드의 데이터를 읽음, 네트워크상의 데이터를 읽음(전송)
- Output : 컴퓨터가 어떤 것을 출력하는 것, 파일에 데이터를 씀, 모니터에 데이터를 씀(출력), 네트워크상에 데이터를 씀(전송)

	이미지동영상 등 데이터용(1byte단위)	문자열용(2byte단위)
입력 API	InputStream	Reader
출력 API	OutputStream	Writer

2) 스트림(Straem) : 데이터를 운반(입출력)하는데 사용되는 연결통로, 하나의 스트림으로 입출력을 동시에 수행할 수 없음(단방향통신), 입출력을 동시에 수행하려면 2개의 스트림이 필요함

- 파일로부터 데이터를 읽는 3단계 : 파일(연결통로)을 연다 -> 파일의 데이터를 읽는다(필요한 만큼 반복) -> 파일을 닫는다

- 파일에 데이터를 쓰는 3단계 : 파일(연결통로)을 연다 -> 파일에 데이터를 쓴다 -> 파일을 닫는다

2-1) InputStream 사용법

1단계 : InputStream(추상) 클래스를 상속받은 여러 가지 API 하위 클래스 중의 하나를 이용해서 객체를 만듬. 또는 다른 클래스의 메소드에서 반환(리턴)되는 타입 객체를 얻음.
2단계 : read(), read(byte[]) 두 개의 메소드를 이용하여 데이터를 읽는다.
- read() 1byte씩 읽는다. 속도가 느리다
- read(byte[]) byte[]만큼씩 읽는다. 속도가 빠르다.
3단계 : 예외 처리와 무조건 close() 실행
- I/O를 하면서 반드시 해야 하는 예외처리가 있음(IOException), 반드시 하도록 컴파일러가 강요
- I/O 작업 마지막은 close()로 외부 연결을 끝내야 함

2-2) OutputStream 사용법

1단계 : OutputStream(추상) 클래스를 상속받은 여러가지 API 하위 클래스들 중의 하나를 이용해서 객체를 만듬. 또는 다른 클래스의 메소드에서 반환(리턴)되는 타입 객체를 얻음.
2단계 : write()메소드를 이용해서 데이터를 읽음
3단계 : write(int), write(byte[]), write(byte[], int, int) 세개의 메소드를 이용 가능.
4단계 : 예외 처리와 무조건 close() 실행
- I/O를 하면서 반드시 해야 하는 예외처리가 있음(IOException), 반드시 하도록 컴파일러가 강요
- I/O 작업 마지막은 close()로 외부 연결을 끝내야 함

3) 파일 읽고 쓰기 : 읽고, 쓰기를 동시에 → 파일 복사
- 파일을 읽고, 다른 파일에 쓰고, 결국은 파일 복사를 뜻함
- 작업순서 : InputStream, OutputStream 준비 → is로 읽어들인 데이터를 os으로 씀 → 외부연결 close()함.
- read() : read(), read(byte[]) 등 다양한 메소드 이용 가능
- write() : write(byte), write(byte[]), (write(byte[], int, int) 등 다양한 메소드 이용 가능

4) File 클래스 : 파일 크기, 속성, 파일이름 정보를 갖고, 생성 및 삭제 메서드 포함
- 파일 안엔 디렉토리도 포함 : 생성, 디렉토리에 포함된 파일 리스트도 가져올 수 있음

4-1) File 클래스 생성/삭제 메서드
- File file = new File("경로명/파일명"); 논리적인 파일이나 디렉토리
- exists(); 현재 파일이나 디렉토리가 있는지 여부
- delete(); 파일 또는 디렉토리 삭제

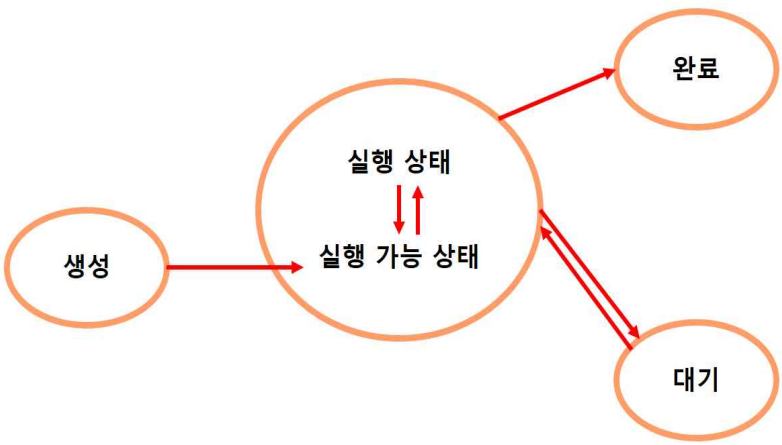
4-2) File 클래스 정보 메서드
- canExecute(); 실행할 수 있는 파일인지 여부
- getName(); 파일이름
- getPath(); 전체 경로
- isFile(), isDirectory(); 파일/디렉토리 인지 여부
- length(); 파일 크기
- list(); 디렉토리인 경우 포함된 파일의 문자열 배열
- listFiles(); 디렉토리인 경우 포함된 파일을 배열로

Day20 Java Thread

1) 스레드 : 하나의 프로그램이 동시에 여러 개의 일을 수행할 수 있도록 해 주는 것

장점	자원을 보다 효율적으로 사용 가능 사용자에 대한 응답성 향상 작업이 분리되어 코드가 간결해짐
단점	동기화에 주의해야 함 교착상태가 발생하지 않도록 주이, 각 스레드가 효율적으로 고르게 실행 될 수 있게 해야함

2) 스레드의 생명 주기 : 생성 > 실행 > 대기 > 완료



3) Synchronized : 먼저 수행되는 스레드의 모든 작업이 끝날 때까지 다른 스레드는 기다리도록 하는 예약어

Day21 Java GUI

1) Java GUI : 아이콘처럼 사용자가 편리하게 그래픽 요소를 가미하여 컴퓨터를 사용하기 쉽게 만들어 놓은 것

2) AWT : JAVA에서 그래픽 요소를 만들기 위한 컴포넌트들.
- Java.awt 패키지 : GUI 구축을 위한 클래스들의 모임

3) Java.awt 패키지에서 제공되는 클래스들의 유형별 분류
3-1) 컴포넌트의 배치와 관련된 클래스 : BorderLayout, GridLayout, FlowLayout

<table>
<tr><td colspan="3">북쪽(NORTH)</td></tr>
<tr><td>서쪽
(WEST)</td><td>중앙
(CENTER)</td><td>동쪽
(EAST)</td></tr>
<tr><td colspan="3">남쪽(SOUTH)</td></tr>
</table>

3-2) GUI 구성과 관련된 클래스 : Button, Label, Canvas, Checkbox, Choice, Scrollbar, List, Menu, TextComponent, TextArea, TextField, CheckboxGroup

3-3) 그래픽 출력과 관련된 클래스 : Color, Font, Rectable, Point, Graphics, Image

3-4) 컨테이너 클래스 그 외의 클래스 : Frame, Panel, Window, Container, FileDialog, Demension, Event

4) AWT 사용법 : 컴포넌트 생성(1. 객체선언-컴포넌트들 부착 2. 객체생성) → 이벤트 처리

5) Swing 컴포넌트 : AWT보다 진보된 기능
- 배치과정
1단계 : 패널또는 컨테이너에 container로 얻어오고(getContentPane()), 레이아웃 셋팅
2단계 : 컴포넌트 추가 및 크기 확인
3단계 : 컴포넌트 add
4단계 : 패널의 크기 확인

Day22 JDBC

1) JDBC : 자바 패키지의 일부로 자바 프로그램이 데이터 베이스와 연결되어 데이터를 주고 받을 수 있게 해 주는 프로그래밍 인터페이스, 자바 데이터베이스 프로그래밍 API

2) JDBC프로그래밍 단계와 사용 클래스

1. JDBC 드라이버 로드 (JDBC 드라이버 로드) : Class.forName()을 이용. 드라이버 클래스 로딩(드라이버 필요)

사용 API : Class.ForName()

2. 데이터베이스 연결 (DB에 연결) : DriverManager.getConnection()을 이용해 Connection 객체 생성(접속 URL, id, passwd등 필요)

사용 API : java.sql.Connection

3. Statement 생성 (SQL문을 수행할 객체 생성) : SQL을 이용해 DB를 조회하거나 다룸. Statement는 SQL 처리 기본 객체

사용 API : java.sql.Statement java.sql.preparedStatement

4. SQL문 전송 (SQL문 수행전송하고 결과 받음): SQL 문 Statement 객체의 executeQuery(), executeUpdate() 메소드를 이용, 데이텁이스로 전달해 처리

사용 API : java.sql.Statement.executeQuery() java.sql.Statement.executeUpdate()

5. 결과 받기 : SQL 실행 결과를 반환 받아야 하는 값이 있다면 ResultSet객체로 받음

사용 API : java.sql.ResultSet

6. 결과 받아 원하는 로직 수행 (수행한 결과를 읽어 원하는 로직 수행(필요할 때까지 반복))

7. 연결 해제 (DB 연결을 끊음) : 사용한 자원을 반납함

사용 API : java.sql.Connection.close()

3) JDBC소스

1. 조회

- Statement객체 : 이전 단계에서 생성한 Connection 객체(conn)로 접근 > createStatement()메소드를 호출하여 생성

// Statement stmt = conn.createStatement();

- Statement객체로 질의문 수행

// String str = "SELECT * FROM EMP;

// ResultSet rs= stmt.excuteQuery(str);

- PreparedStatement 인터페이스 이용
String sql_query = "insert into dept values (?,?,?);
PreparedStatement pstmt = conn.preparedStatement(sql_query);
setXXX(int 순서, 실제 데이터나 변수);

2. 삽입
- Stmt.excuteQuery(sql) : 검색(Select)시 사용 반환값이 ResultSet
- Stmt.excuteUpdate(sql) : 입력, 수정, 삭제(insert, update, delete)시 사용.
반환 값이 정수형을 반환. 작업에 성공한 횟수 리턴하여 DML명령어(insert, update, delete)가 제대로 수행되었는지 체크