- JAVA는 OS 독립적 언어. 가상머신 통해 여러 OS에서 실행 가능
<-> CPU 종속적 언어 ex:어셈블리, OS 종속적 언어 ex: C, C++

- #가상머신
Java Virtual Machine. JVM. OS독립적, 객체지향 언어. 개발과 유지보수가 효율적. 단점은 속도저하.

- JAVA #SE(Stand Edition) : Java Application 개발. 데스크탑에서 실행되는 응용PG 개발

- JAVA ME(Mobile Edition) : 휴대 가능한 소형 디바이스에 사용을 목적으로 개발. 현재는 쇠퇴

- JAVA EE(Enterprise Edition) : 기업 솔루션, 서버와 클라이언트 PG 개발 목적

- JAVA 언어의 특징
1) Virtual Machine(가상머신) : 운영체제의 영향을 받지 않는다.
2) Garbage Collection : JVM이 메모리 자동관리 (개발자는 메모리 관리를 하지 않는다.)
3) Object Oriented Program(OOP) : 객체지향 언어로 개발과 유지보수가 효율적
4) 분산 네트워크 기술을 지원 : 원격 메소드 호출(Remote Method Invocation, RMI), 네트워크 상 다른 computer에 있는 서버를 Java 프로그래밍으로 연결해서 처리 가능(Common Request Broker Architecture, CORBA)
5) 다중 쓰레드를 지원한다. (네트워크 상 한 번에 여러 클라이언트 접속, 하나의 프로세스 안에 여러 단위의 요청 내용을 처리)
6) 보안 기능 지원 - 접근에 대한 권한을 변수로 처리 - Access Modifier(접근제어자)

- #JDK : Java Development Kit : 자바 개발 키트

- #IDE : Integrated Development Environment : 통합 개발 툴

- bin/javac.exe(컴파일러) : *.java 파일을 JVM이 받아들일  수 있는 .class 파일로 변환시켜주는 프로그램.

- eclipse에서 workspace는 어떤 역할을 하는가? 작업 환경이 저장된다.

- dot(.)는 접근 연산자. 예) System.out.println => System 이라는 JRE 안에 라는 뜻.

- out : 출력에 관한 부분 제공

- #변수(variable)?
1) 프로그램 작업을 처리하기 위해 하나의 값을 저장할 수 있는 메모리 공간
2) 임의의 메모리 공간에 이름을 붙여 관리하는 것
3) Java에서는 다양한 타입을 저장할 수 없고, 한 가지 타입만 값으로 저장할 수 있다.
4) 식별자(identifier)라고도 한다. 자바 코드에서 변수로 입력시킨 이름.

- 기본 자료형

| 자료형 | 키워드 | 크기 | 표현범위 |
|---|---|---|---|
| 논리형 | boolean | 1 byte | true, false |
| 문자형 | char | 2 byte | 0~65,535 |
| 정수형 | byte | 1 byte | -128 ~ 127 (-27~ 27-1) |
| | short | 2 byte | -32,768 ~ 32,767 (-215~ 215-1) |
| | **int** | 4 byte | -2,147,483,648 ~ 2,147,483,647 |
| | long | 8 byte | -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 |
| 실수형 | float | 4 byte | -3.4E38 ~ +3.4E38 |
| | **double** | 8 byte | 1.7E308 ~ + 1.7E308 |

- 기본 데이터 타입 VS 참조 데이터 타입

1) 기본 데이터 타입(primitive 기본 자료형) :

메모리에 있는 실제 값 = 변수 데이터 값. Java 언어에 이미 존재하고 있는 데이터 타입.

2) 참조 데이터 타입(object, 객체 자료형) :

메모리에 있는 실제 값(stack 영역) = 변수가 저장된 주소(실제 변수 데이터는 heap 영역)

*stack 영역

- heap 영역에 생성된 Object 타입의 데이터의 참조값이 할당된다.
- 원시 타입(primitive types)의 데이터가 값과 함께 할당된다.
- 지역변수들은 scope에 따른 visibility를 가진다.
- 각 Thread는 자신만의 stack을 가진다. (각 스레드에서 다른 스레드의 stack 영역에는 접근할 수 없다.)

*heap 영역

- 주로 긴 생명주기를 갖는 데이터 저장
- 애플리케이션의 모든 메모리 중 stack에 있는 데이터를 제외한 부분
- 모든 Object 타입(Integer, String, ArrayList...)은 heap 영역에 생성된다.
- 몇 개의 스레드가 존재하든 상관 없이 단 하나의 heap 영역만 존재한다.
- heap 영역에 있는 오브젝트들을 가리키는 레퍼런스 변수가 stack 영역에 올라간다.

- JVM의 Garbage Collector는 Unreachable Object를 우선적으로 메모리에서 제거하여 메모리 공간을 확보한다. Unreachable Object = Stack에서 도달할 수 없는 Heap 영역의 객체. Garbage Collection이 일어나면 Unreachable 오브젝트들은 메모리에서 제거된다. Garbage Collection = Mark and Sweep. Garbage가 아닌 것을 따로 mark 하고 그 이외의 것을 지우는 것.

- 형변환

1) 묵시적 변환 :

int i1 = 10;

long l1 = 220000L;

double d1 = i1;

2) 명시적 변환 : 데이터 손실이 있을 수 있으며 코드에 명시해주어야 한다.

double d2 = 10.1;

int i2 = (int)d2;

- 연산자

(우선순위에 따른 종류)

1) 1차 연산자 : ( )

2) 단항연산자_증감연산자 : !, ++, -- (우결합성)

3) 이항연산자_산술연산_승법연산자 : *, /, %

4) 이항연산자_산술연산_가법연산자 : +, -

5) 이항연산자_관계연산자 : <, <=, >, >=, ==, !=

6) 이항연산자_비트연산자 : &, |

7) 이항연산자_논리연산자 : &&, ||

8) 조건연산자(삼항연산자) : ? :

9) 할당연산자 : =, +=, -=, *=, /=, %= (우결합성)

- 반복문

1) for 문 : 미리 설정된 횟수만큼 반복적으로 수행

for(초기값; 반복할 조건; 증감식){ 반복할명령문; }

```
public class Ex1{
     // 1부터 20까지 누적 합 구하기
     public static void main(String[] args){
          int total = 0;
          for(int i; i<=20; i++){
               total += i;
          }
          Sytem.out.println(total);
     }
}
```

2) while문 : 조건이 만족될 때까지 반복적으로 수행
while(조건식){ 조건식이 참일 때 계속 실행할 명령문; }

```
public static void main(String[] args){
     int i = 0;
     while(i<=10){
          System.out.println("i="+(i++));
     }
}
```

3) do-while문 : 우선 수행한 후 조건이 만족되면 수행, 만족되지 않으면 수행하지 않는다.
do{
    최초 한 번은 무조건 실행=무조건 실행될 명령문
}while(조건식){
    조건식이 참인 경우 수행할 명령문
}

```
public static void main(String[] args){
     int num;
     Scanner sc = new Scanner(System.in);
     do{
          System.out.print("한 자리 짝수를 입력하세요");
          num = sc.nextInt();
     }while(num%2==1){
          System.out.print("홀수에여");
     }
}
```

- #배열
동일 자료형의 집합. 하나의 이름으로 다수 개의 데이터 사용 가능. 여러 개의 데이터를 가르키고 있는 주머니. 즉, 여러개의 변수를 모아 놓은 또 하나의 주머니라고 생각하면 쉽다. 배열의 크기는 최초에 한 번 설정되면 변경이 불가하다. 배열은 ㄱ개체로 취급. 배열을 선언 -> 배열의 메모리 할당(배열 생성) -> 배열 이용

```
// 배열 선언
int [] score;
// 메모리 확보
score = new int[3];
// 값할당
score[0] = 1000;
score[1] = 2000;
score[2] = 3000;
```

- 레퍼런스
배열을 구성하고 있는 데이터들의 주소값. 동일한 레퍼런스(주소값)를 갖고 있다면 같은 데이터.

```java
package com.practice.java;

import java.util.Scanner;

public class Array {
    public static void main(String[] args) {
        String[] friend = {"친구1", "친구2", "친구3", "친구4", "친구5"};
        int[] height = new int[5];
        int totHeight = 0;
        Scanner sc = new Scanner(System.in);
        for(int i=0; i<height.length; i++) {
            System.out.println(friend[i]+"의 키는?");
            height[i]=sc.nextInt();
            totHeight += height[i];
        }
        System.out.print("평균키 : ");
        System.out.println(totHeight/height.length+"cm");
        sc.close();

        int maxH = 0;
        int maxIndex = 0;
        for(int i=0; i<height.length; i++) {
            if(height[i]>maxH) {
                maxH = height[i];
                maxIndex = i;
            }
        }
        System.out.println("가장 큰 학생 : "+friend[maxIndex]+"("+height[maxIndex]+"cm)");
    }
}
```

- 배열의 복사

1) for문 이용

```java
package com.practice.java;

public class Array2 {
    public static void main(String[] args) {
        int[] num = {0,1,2,3,4};
        int[] newNum = new int[5];
        for(int i=0; i<newNum.length; i++) {
            newNum[i] = num[i];
        }
        for(int i
                =0; i<newNum.length; i++) {
            if(i==0) {
                System.out.print("int[] newNum = {");
            }else if(0<i && i<=newNum.length-3) {
                System.out.print(newNum[i]+", ");
            }else if(i==newNum.length-2) {
                System.out.print(newNum[i]+", ");
            }else if(i==newNum.length-1) {
                System.out.print(newNum[i]+"}");
            }
        }
    }
}
```

2) System.arraycopy 이용

System.arraycopy(원본배열객체, int 원본 시작 위치, 복사본 배열 객체, int 복사본 시작 위치, int 복사 길이)

```
package com.practice.java;

public class Array2 {
    public static void main(String[] args) {
        int[] num = {0,1,2,3,4};
        int[] newNum = new int[5];
        /*
         * for(int i=0; i<newNum.length; i++) { newNum[i] = num[i]; }
         */
        System.arraycopy(num, 0, newNum, 0, num.length);
        for(int i
                =0; i<newNum.length; i++) {
            if(i==0) {
                System.out.print("int[] newNum = {");
            }else if(0<i && i<=newNum.length-3) {
                System.out.print(newNum[i]+", ");
            }else if(i==newNum.length-2) {
                System.out.print(newNum[i]+", ");
            }else if(i==newNum.length-1) {
                System.out.print(newNum[i]+"}");
            }
        }
    }

}
```

- 다차원배열
int[][] array = new int[i][j]
i : 다차원 배열의 행 수(=가질 수 있는 내부 배열의 개수)
j: 다차원 배열의 내부 배열(?)이 가질 수 있는 객체의 수
예) int[][] array = new int[3][5]
array = {{1,2,3,4,5},  {2,3,4,5,6}, {3,4,5,6,7}}
array[0][1] = 1;

- 절차 지향 : 위에서부터 순차적으로 실행
- 절차 지향의 단점 :
1) 동일 작업인데 다시 코드 수정을 해야하는 경우가 있음.
2) 동일한 로직의 코드가 문서 내에 넘쳐나 코드가 길어짐.
3) 유지보수가 어려움.
=> 그래서 등장한 방식이 함수 또는 메소드(method)
로직만 만들어 놓고, 그 때 그 때 데이터를 주면 메소드가 알아서 결과값을 반환.

- 객체지향
객체 : 동일한 성질의 데이터와 메소드를 한 곳에 모아두고 필요한 곳에서 언제든지 이용할 수 있게 만들어 놓은 덩어리. 이런 객체를 이용한 프로그래밍 방식이 객체지향 방식.

```java
package com.practice.java;

public class Oop {
    public String evenOdd(int value) {
        String result = null;
        if(value%2==0) {
            result = "짝수입니다.";
        }else {
            result = "홀수입니다.";
        }
        return result;
    }
    public int sum(int from, int to) {
        int result = 0;
        for(int i=from; i<=to; i++) {
            result +=i;
        }
        return result;
    }
    public int abs(int su) {
        int result = (su<0)? -su:su;
        return result;
    }
}
```

```java
package com.practice.java;

public class ExMain {
    public static void main(String[] args) {
        Oop oop = new Oop();
        int sum = oop.sum(1, 10);
        System.out.println("1부터 10까지 정수의 합 : "+sum);
        System.out.println(oop.evenOdd(sum));
        System.out.println("-10의 절대값은 : "+oop.abs(-10));
    }
}
```

- #메소드 : 작업을 수행하기 위한 명령문의 집합
어떤 값을 입력받아서 처리하고 그 결과를 돌려준다. (입력 받는 값이 없을 수도 있고, 결과를 돌려주지 않을 수도 있다. <- void) 반복적으로 수행되는 여러 문장을 메소드로 작성한다.

- 메소드 작성지침
접근제한자(static) 리턴type(int, String....) 메소드명(매개변수1, 매개변수2, ...){
// 리턴이 없는 경우 리턴 type은 void
처리할 프로세스들
return 리턴값;
}

- 재귀적 호출
자기자신 즉 실행되고 있는 메소드 내부에서 메소드 자신을 다시 호출하는 것. 반복문으로 대체 가능. 재귀호출은 반복문보다 성능이 나쁘다.

```java
public class Factorial {
    private static int factorial(int s) {
//      int fact = 1;
//      for(int i=s; i>=1; i--) {
//          fact = fact*i;
//      }
//      return fact;
        return (s==1)? 1:s*factorial(s-1);
    }
```

- 객체의 개념 및 클래스의 이해
객체 = 자동차. 데이터(속성) = 색상, 배기량, 속도. 메소드 = 드라이브, 주차, 레이싱. 객체는 메소드의 상위 개념

- 객체 모델링 : 현실세계나 추상적인 내용의 속성과 동작을 추려내어 소프트웨어 객체의 필드와 메서드로 정의해 나가는 과정

- 객체는 클래스로부터 메모리에 생성된다. 클래스 정의 -> 클래스로부터 객체 생성. 프로그램에서의 객체 = 데이터+메소드.

- 생성된 객체는 동일한 클래스에서 생성되었다 하더라도 완전한 독립체. 즉, 객체 안에 종속되어 있는 데이터는 완전히 별개의 데이터.

- 캡슐화(Encapsulation) : 객체에 포함된 속성과 메소드를 객체간에 감추거나, 권한에 따라 접근이 가능하게 처리하는 것. 키워드는 접근제어자(access modifier)

- 클래스 제작
1) 패키지명
2) 클래스명
3) 데이터
4) 생성자함수
5) 메소드
6) getter, setter

```java
package com.practice.java; // 패키지명

public class Square { // 클래스명
    // 데이터
    private int side; // 자동으로 0 초기화
    // 생성자 함수 (*아무런 생성자도 만들지 않으면, 자동으로 매개변수 없는 생성자만 생긴 것이나 다름없다.)
    public Square() { // 매개 변수 없는 생성자
    }
    public Square(int side) { // 매개 변수 있는 생성자
        this.side = side;
    }
    // 메소드
    public int area() {
        return side*side;
    }
    // getter
    public int getSide() {
        return side;
    }
    // setter
    public void setSide(int side) {
        this.side = side;
    }


}
```

```java
package com.practice.java;

public class SquareMain {
    public static void main(String[] args) {
        Square s1 = new Square(5);
        Square s2 = new Square(25);

        System.out.println("s1의 넓이 : "+s1.area());
        System.out.println("s2의 넓이 : "+s2.area());
    }
}
```

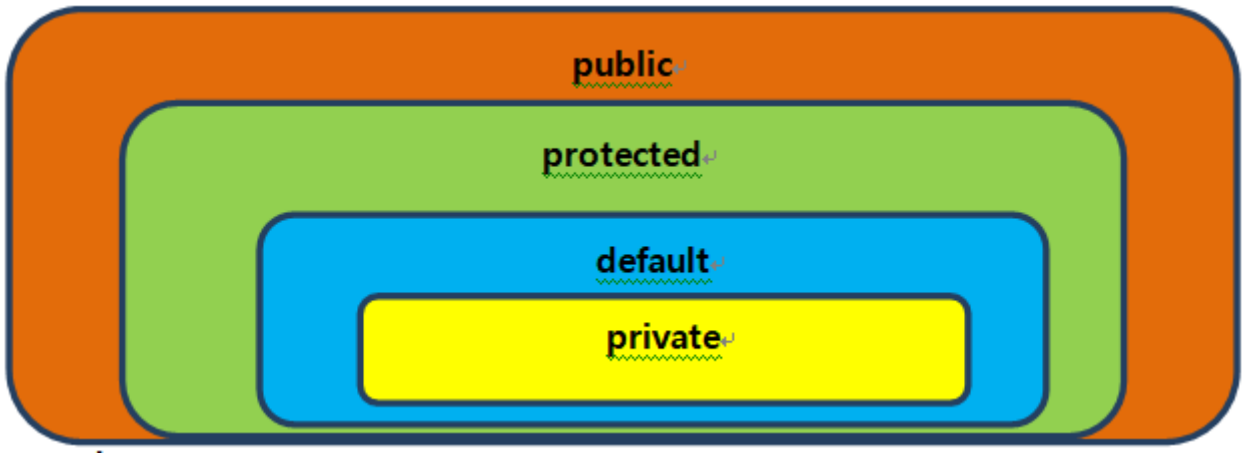
- 접근제한자 : 접근 제한이란 클래스의 데이터나 메소드에 대해서 다른 클래스로부터 접근을 제한 하는 것.

1) public : 접근을 제한하지 않는다. 다른 모든 클래스에서 사용 가능

2) protected : 해당 클래스와 동일한 패키지에 있거나 상속받은 클래스일 경우에만 사용 가능

3) default : 접근 제어자를 명시하지 않은 경우. 같은 패키지 내의 클래스들은 public 권한을 갖고 접근 가능

4) private : 해당 클래스만이 이 멤버를 사용할 수 있다. 외부 객체에서는 절대로 접근할 수 없다.




- static/final

1) final : 종단변수. 상수. 값을 변경하지 못한다.

2) static : 클래스 변수. static 변수. 객체 변수(객체 속성)는 객체가 생성될 때 마다 각 객체 안의 속성 변수들이 생성되지만, 클래스 변수는 클래스로부터 생성된 객체들의 수와 상관없이 하나만 생성된다. 한 클래스로부터 생성된 모든 객체들은 클래스 변수를 공유. 클래스 이름을 통해 접근.

- static의 장점과 단점

객체 생성을 하지 않고 사용가능한 것은 장점. but Garbage Collector의 관리 밖에 있기 때문에 항상 메모리에 상주. 따라서 신중히 해야.

- 멤버 변수의 초기화 시기와 순서

1) 클래스 변수의 초기화 시점 : 클래스가 처음 로딩될 때 딱 한 번

2 인스턴스 변수의 초기화 시점 : 인스턴스(객체)가 생성될 때 마다

- 패키지

프로그래밍에서 여러 클래스를 관리하기 위해 기능적으로 영향을 미칠 수 있는 클래스끼리 묶어 놓고, 접근 범위 안에 효과적으로 호출하기 위해서 사용하는 개념이다. 패키지 이름은 유일한 이름이여야 한다. 주로 유니크한 도메인 주소를 역방향으로 하여 만든다. 명명 규칙은, 숫자로 시작할 수 없고 _나 $ 외의 특수문자를 쓸 수 없다. 그리고 모두 소문자로 작성한다.

- 상속

다른 클래스로부터 데이터(속성)와 메소드(기능)를 이용하거나 변경할 수 있도록 하는 것. 상속 받는 것-. 상속을 통해 결과물을 빠르게 만들 수 있다. (기존 프로그램은 버그도 거의 없을 것). 다양한 객체(타입)를 상속을 통해 하나의 객체(타입)로 묶을 수 있다. 기존의 클래스를 재 사용해서 새로운 클래스를 작성하는 것. 두 클래스를 부모와 자식(조상과 자손) 관계로 맺어줄 수 있다. 자손은 조상의 멤버변수를 상속받지만, private 멤버변수는 직접 제어할 수 없다. 자손의 멤버 개수가 조상보다 적을 수는 없다. (같거나 더 많다.) Java에서는 다중상속은 지원되지 않는다.

- 오버로딩

overloading = polymorphism 중복정의. 인자의 타입이 다르면 같은 이름의 메소드라도 다른 기능으로 중복 정의가 가능하다.

- 오버라이딩

overriding. 부모 클래스의 메소드를 자식 클래스에서 재정의 함. 자식 클래스가 부모 클래스를 상속하여 자식한테 없는 메소드를 호출하면 부모 클래스에 가서 해당 메소드를 찾는다. 만약 부모 클래스의 메소드를 자식 클래스에서 동일한 이름으로 다시 재정의 하면 자식 클래스의 메소드를 호출. 이게 오버라이딩.

- 다형성
같은 형태지만 기능적으로 여러 다양한 객체를 이용할 수 있는 성질

- 추상클래스
추상적으로 정의할테니 (=선언 해놓을테니) 객체를 사용할 사용자가 꼭 재정의(Overrideing) 하세요~

- 추상클래스 문법
public abstract class ClassName(){ }
추상클래스에는 하나 이상의 메소드 포함. 추상 메소드에는 선언부만 있고 실행부는 없다. 추상클래스는 메소드 선언만 하고 실제 구현은 상속받는 클래스에서 한다. 클래스의 프레임만 구성하는 셈. 직접 객체 생성이 불가능하다. abstract = 인스턴스화를 금지.

- 인터페이스
작업명세서. 실제 구현된 것은 전혀 없는 기본 설계도. 일종의 추상 클래스. 추상 클래스보다 추상화 정도가 높다. 인스턴스를 생성할 수 없고, 클래스 작성에 도움을 줄 목적으로 사용된다. 미리 정해진 규칙에 맞게 구현하도록 표준을 제시하는 용도. 추상 메소드와 상수만을 가질 수 있다. 다형성을 가능하게 한다. 객체를 부속품화. 패턴이나 프레임워크. 객체와 객체 간의 소통 수단.

- 인터페이스의 문법

```
public interface 인터페이스이름{
     public static final 타입 상수이름 = 값;
     public abstract 메소드이름(매개변수 목록);
}
```

모든 멤버 변수는 public static final 이어야 하며 이를 생략할 수 있다. 모든 메소드는 public abstract 여야 하며 이를 생략할 수 있다. private는 불가. 메소드는 무조건 추상 메소드만 존재. 구현은 implement되는 자식 클래스에서.

- 다형성
여러가지 형태를 가질 수 있는 능력. 객체의 형태가 다양하게 변할 수 있는 것.

- 오버라이딩의 조건
1) 선언부가 같아야 한다.
2) 접근 제어자를 더 좁은 범위로 변경할 수 없다. 예를 들어 조상메소드가 protected라면 protected, public 만 가능

- 오버로딩 vs 오버라이딩
오버로딩은 컴파일러 입장에서 새로운 메소드를 정의하는 것.(new) 메소드 다중정의. 같은 class에서 동일한 메소드가 매개변수를 달리 여러개 존재. 오버라이딩은 상속받은 메소드의 내용을 변경하는 것. 메소드 재정의.

- 다중 상속은 안되지만 다중 구현은 가능하다.

- 인터페이스의 장점
1) 개발 시간을 단축시킬 수 있다. - 인터페이스를 구현하는 클래스가 작성될 때까지 기다리지 않고도 양쪽에서 동시에 개발이 가능하다.
2) 표준화가 가능하다. - 일관되고 정형화된 프로그램 개발 가능
3) 서로 관련없는 클래스들에게 관계를 맺어줄 수 있다. 서로 상속 관계에 있지도 않고, 같은 조상 클래스를 가지고 있지도 않은 서로 아무런 관계도 아닌 클래스들에게 하나의 인터페이스를 공통적으로 구현하도록 함으로써 관계를 맺어줄 수 있다.
4) 독립적인 프로그래밍이 가능하다. - 클래스의 선언과 구현을 분리시킬 수 있기 때문에 실제 구현에 독립적인 프로그램을 작성하는 것이 가능하다.

- 인터페이스와 추상클래스
(공통점)
1) 추상메소드를 갖는다.
2) 변수 타입을 정의하는 것이 목적이다.
3) 객체생성은 anonymous(익명클래스) 이용.

(차이점)
1) 추상메소드는 상속을 통한 사용이고 인터페이스는 구현을 통한 사용
2) 인터페이스에는 상수와 추상메소드만 존재
3) 단일 상속, 다중 구현

- 디자인 패턴
개발자 선배님들이 객체지향 언어의 장점을 모아 가장 효율적으로 개발할 수 있게 만들어 놓은 프레임.

- 싱글톤 패턴
어떤 클래스의 객체는 오직 하나인 유일한 객체를 만들고, 여러가지 상황에서 동일한 객체에 접근하기 위해 만들어진 패턴. 생성자가 여러차례 호출 되더라도 실제로 생성되는 객체는 하나고 이 객체에 접근할 수 있는 전역적인 접촉점을 제공하는 패턴.

- 스트레티지(Strategy) 패턴
기능 하나를 정의하고 각각을 캡슐화하여 교환해서 사용할 수 있도록 한다. 즉 [[[[[[[[[[[[[기능을 부품화]]]]]]]]]]]]], 표준화 하는 것.

- String의 주요 기능
String concat(String str)
String substring(int begin, int end)
int length()
String toUpperCase()
String toLowerCase()
char charAt(int index)
intIndexOf(char ch)
intLastIndexOf(char ch)
String trim()
String replace(char old, char new)
String replaceAll(String old, String new)

- String의 문제점
메모리를 과소비한다. 처음 초기화된 데이터에 변화가 생기면 기존의 것을 재활용 하지 않고 새로운 메모리를 이용한다. 즉, String 객체가 가지고 있는 문자열의 내용을 바꾸는 것이 아니라 바뀐 내용을 갖는 새로운 String 객체를 생성해 낸다. 때문에 문자열 조작을 많이 하는 프로그램에서 많이 사용하면 비효율적이다.

- StringBuffer, StringBuilder
문자열 변수의 잦은 내용 변경이 불가피할 때 적합. 객체 내부에 있는 버퍼에 문자열 내용을 저장해 두고 그 안에서 추가, 수정, 삭제 작업을 한다. 새로운 객체를 만들지 않고도 문자열 조작을 할 수 있다. 속도적인 측면에서 개선.

- StringBuilder의 주요 기능
append(String str)
insert(int index, String str) : 특정 index에 문자열 str 추가
delete(int start, int end) : start 위치부터 end 앞 까지 삭제
deleteCharAt(int index) : index 위치의 특정 문자 삭제
int capacity() : 문자열 크기 반환
ensureCapacity(int size) : 버퍼의 크기를 size만큼 늘림.
trimToSize() : 과도한 버퍼의 크기를 적당하게 줄임.

- System.currentTimeMillis()
1970년도부터 현재까지의 밀리세컨. 속도 테스트 용.

- Calendar

```
Calendar calendar = Calendar.getInstance();
        int year = calendar.get(Calendar.YEAR);
        int month = calendar.get(Calendar.MONTH) + 1; // 0~11월이라현실이랑맞춰줌
        int day = calendar.get(Calendar.DAY_OF_MONTH);
        int week = calendar.get(Calendar.DAY_OF_WEEK);//1:일 2:월 3:화 4:수 5:목 6:금 7:토
        int hour24 = calendar.get(Calendar.HOUR_OF_DAY); // 24시
        int hour12 = calendar.get(Calendar.HOUR); // 12시
        int ampm = calendar.get(Calendar.AM_PM);
        int minute = calendar.get(Calendar.MINUTE);
        int second = calendar.get(Calendar.SECOND);
        int millisec = calendar.get(Calendar.MILLISECOND);
```

- Math

```
//Math의 static method 실습
        System.out.println("2의 3승 : "+Math.pow(2, 3));
        System.out.println("-9.9의절대값:"+Math.abs(-9.9));
        System.out.println("16의제곱근 : "+Math.sqrt(16));
        System.out.println("4와 7중에작은값:"+Math.min(4, 7));
        System.out.println("7과 9중최대값:"+Math.max(7, 9));
//Math의 static final 변수(상수)
        System.out.println("sin(PI)="+Math.sin(Math.PI));
        System.out.println("cos(PI)="+Math.cos(Math.PI));
        System.out.println("tan(PI)="+Math.tan(Math.PI));

System.out.println(" 소숫점에서반올림, 올림, 버림");
        System.out.println("9.12의올림 : "+Math.ceil(9.12));
        System.out.println("9.69의반올림 : "+Math.round(9.69));
        System.out.println("9.69 버림 : "+Math.floor(9.69));
        System.out.println("소숫점한자라에서반올림, 올림, 버림");
        System.out.println("9.12의올림 : "+Math.ceil(9.12*10)/10);
        System.out.println("9.69의반올림 : "+Math.round(9.69*10)/10.0);
        System.out.println("9.69 버림 : "+Math.floor(9.69*10)/10);
        System.out.println("십의자리에서반올림, 올림, 버림");
        System.out.println("11의올림 : "+Math.ceil(11/10.0)*10);
        System.out.println("19의반올림 : "+Math.round(19/10.0)*10);
        System.out.println("19 버림 : "+Math.floor(19/10.0)*10);
```

- random

```
publicstaticvoid main(String[] args) {
        Random random = new Random();
        System.out.println("0부터 100까지의난수 : " + random.nextInt(101));
        System.out.println("0부터 50까지의난수발생 :" + random.nextInt(51));
        System.out.println("0부터 20까지의난수발생 : " + random.nextInt(21));
        System.out.println("int형전체범위의난수발생 : " + random.nextInt());
        System.out.println("float 타입의난수발생 : " + random.nextFloat());
        System.out.println("double 타입의난수발생 : " + random.nextDouble());
        System.out.println("long 타입의난수발생 : " + random.nextLong());
        System.out.println("boolean타입의난수발생 : " + random.nextBoolean());
byte[] bytes = newbyte[5];
        random.nextBytes(bytes);
        for(inti=0; i<bytes.length ; i++) {
            bytes[i] = (byte)Math.abs((byte) ((bytes[i]%45)+1));
            System.out.println(bytes[i]);
        }
    }
```

- Wrapper 클래스

기초데이터를 객체 데이터로 변환.

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

```
class WrapperExample1 {
public static void main(String args[]) {
        Integer obj1 = new Integer(12);
        Integer obj2 = new Integer(7);
int sum = obj1.intValue() + obj2.intValue();
System.out.println(sum);
    }
}
```

```
※String을 기초데이터로 반환하는 메소드들
    Byte.parseByte("1");
    Short.parseShort("23");
    Integer.parseInt("123456");
    Long.parseLong("123456");
    Float.parseFloat("1.5");
    Double.parseDouble("1.00005");
    Boolean.parseBoolean("true")
    (反) String.valueOf(1) =>“1”
```

- 예외 처리 문법

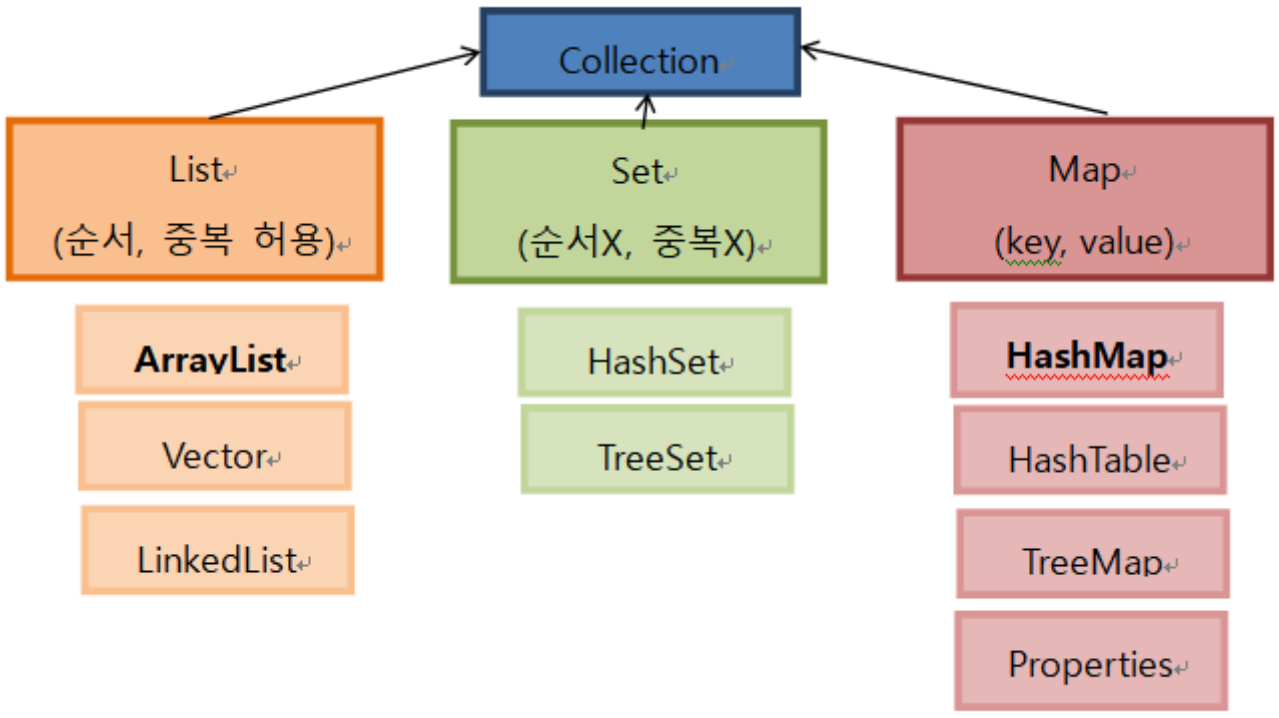
```
try {
    try블럭 ;익셉션이 발생할 가능성이 있는 명령문들(문제가 발생할 수 있는 로직을 기술)

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

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

- Collection

자료구조. 다수의 데이터를 쉽게 처리할 수 있는 방법




- ArrayList
- LinkedList
- Vector
- HashMap
- HashSet
- Iterator

| | 특 징 |
|---|---|
| List | 순서가 있는 데이터의 집합, 데이터의 중복을 허용한다<br>ex. 대기자명단 |
| | 구현클래스 : ArrayList, LinkedList, 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<br>순서는 유지되지 않으며, 키는 중복을 허용하지 않고 값은 중복을 허용한다 |
| | 구현클래스 : HashMap, HashTable등 |

- TreeSet

- InputStream : 1byte 단위 입력 API. 이미지, 동영상 등 데이터에 주로 사용

- ouputStream : 1byte 단위 출력 API. 이미지, 동영상 등 데이터에 주로 사용

- Reader : 2byte 단위 입력 API. 문자열에 주로 사용

- Writer : 2byte 단위 출력 API.  문자열에 주로 사용

- DataInputStream / DataOutputStream

- BufferedReader / BufferedWriter

- PrintWriter

- 하나의 스트림에 입출력을 동시에 수행할 수는 없다. 단방향 통신. 입출력을 동시에 수행하려면 2개의 스트림이 필요하다.

```java
package com.practice.java;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class OuptutStream {
    public static void main(String[] args) {
        OutputStream os = null;
        try {
            os = new FileOutputStream("D:/Java/outTest.txt");
            String temp = "Hello JAVA!";
            byte[] bs = temp.getBytes();
            for(int idx=0; idx<bs.length; idx++) {
                System.out.println(bs[idx]);
                os.write(bs[idx]);
            }
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage()+"파일만들기 오류");
        } catch (IOException e) {
            System.out.println(e.getMessage()+"내용 쓰기 오류");
        } finally {
            try {
                if(os!=null) os.close();
            }catch(Exception e) {

            }
        }
    }
}
```

- File 클래스
파일 크기, 속성, 파일이름 정보를 갖고 생성 및 삭제 메소들르 포함한다. 생성, 디렉토리에 포함된 파일리스트도 가져올 수 있다.

- 생성/삭제 메소드
File file = new File("경로명/파일명"); 논리적인 파일이나 디렉토리
exist(); 현재 파일이나 디렉토리가 있는지 여부
delete(); 파일 또는 디렉토리 삭제

- 정보 메소드
canExcute(); 실행할 수 있는 파일인지 여부
getName(); 파일 이름
getPath(); 전체 경로
getAbsolutePath(); 절대경로
getCanonicalPath(); 표준경로
isFile(), isDirectory(); 파일/디렉토리 인지 여부
length(); 파일 크기
list(); 디렉토리인 경우 포함된 파일의 문자열 배열
listFiles(); 디렉토리인 경우 포함된 파일을 배열로
lastModified(); 최종 수정 시간
Date date = new Date(file.lastModified());

- SimpleDateFormat sdf = new SimpleDateFormat("yyyy년 MM월 dd일 E a h시 m분 s초");
  sdf.format(시간);

- 프로세스

프로그램을 실행하면, 프로그램이 메모리에 상주한다. 이게 프로세스. CPU가 동시에 여러 프로그램을 실행하는 것이 멀티 프로세스. 하지만 CPU는 하나이기 때문에 실행 시간을 잘게 쪼개 프로세스가 돌아가는 것. 각 프로세스가 돌아가면서 CPU를 점유. 프로세스 간 CPU점유 작업이 스케줄링.

- 스레드
하나의 프로그램이 동시에 여러 개의 일을 할 수 있도록 해주는 것. 하나의 새로운 프로세스를 생성하는 것보다 하나의 새로운 쓰레드를 생성하는 것이 더 적은 비용이 든다. 스레드는 두 가지 형태로 존재한다. 하나는 n개의 스레드가 객체 하나를 공유하는 방식. 또 다른 하나는 n개의 스레드가 각각 n개의 객체를 사용하는 방식이다.

- 멀티 스레드의 장점
1) 자원을 효율적으로 사용
2) 사용자에 대한 응답성 향상
3) 작업이 분리되어 코드가 간결

- 멀티 스레드의 단점
1) 동기화에 주의
2) 교착상태(dead-lock)가 발생하지 않도록 주의
3) 각 쓰레드가 효율적으로 고르게 실행될 수 있도록 한다.

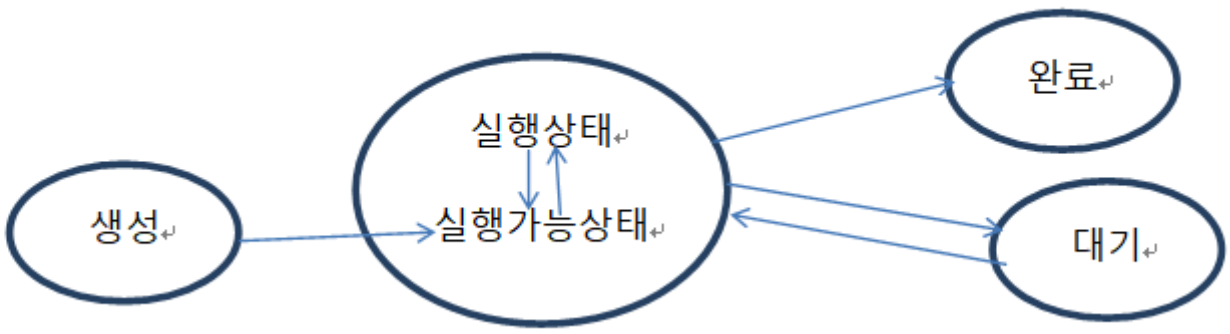
스레드의 생명주기

가) 생성

나) 실행

다) 대기

라) 완료




- synchronized
먼저 수행되는 스레드의 모든 작업이 끝날 때 까지 다른 스레드를 기다리도록 하는 것.

4. 데이터베이스의 특성(DB를 사용해야 하는 이유)
   - Realtime accessibilities(실시간처리) 생성된 데이터를 즉시 컴퓨터로 보내 처리하는 방식. 질의에 대한 실시간 처리 및 응답. (원할 때마다 쓰고 읽는다)
   - Continuous Evolution(계속 변화) 새로운 데이터의 insert, delete, update 등의 기능이 수시로 이루어진다
   - Concurrent Sharing(공유) 여러 사용자가 자기가 원하는 데이터에 동시에 접근하여 사용가능
   - Content Reference(내용에 의한 참조) 데이터 레코드들의 주소나 위치가 아니라 사용자가 요구하는 내용, 즉 데이터가 가지고 있는 값에 따라 참조된다

데이터베이스 관리 시스템의 장점

- 데이터 중복의 최소화
  - 데이터를 통합하여 관리하므로 데이터의 중복 제어 가능
- 데이터 공유
  - 데이터의 통합 관리를 위해 데이터를 공통으로 사용할 수 있도록 데이터를 공통의 저장소에 저장하고 이를 이용하여 데이터를 사용하도록 함
- 데이터의무결성, 일관성 유지
  - 데이터가 중복을 제거하고 데이터의 공유함으로써 데이터간의 불일치가 발생하지 않도록 하여 데이터 관리의 일관성 유지.
  - 데이터베이스에 저장된 데이터 값과 실제 값이 일치하도록 함으로써 무결성 유지.
- 데이터의 보안 보장
  - 데이터베이스를 중앙집중식으로 관리하기 하기 때문에 데이터베이스의 관리 및 접근을 효율적으로 관리함으로 모든 데이터에 대해 보안 제공.
- 데이터 관리 표준화 (업무의 표준화가 가능)
  - 데이터가 의미하는 내용과 표현하는 형태 사이의 불일치를 방지하기 위해 데이터들에 대한 기준을 명확히 하고 동일한 항목들에 대해 같은 기준이 적용될 수 있도록 함.
- 데이터 관리의 유연성
  - 새로운 데이터에 대한 요구에 대해 유연하게 대처할 수 있도록 하는 것을 의미.
  - 모든 데이터를 공유하여 관리함으로써 사용자가 사용하지 않는 데이터들을 공용으로 사용할 수 있도록 함으로써 새로운 데이터 요구에 대하여 유연하게 대응.

데이터베이스 관리 시스템의 단점

- 운영비가 많이 든다 ; 대용량 메모리와 고속 CPU 요구 등의 초기 운영비, 유지보수비, 다양한 요구를 충족시키기 위한 개발비
- 자료처리 방법이 복잡해 질 수 있다 ; 상이한 데이터가 상호 관련되어 많은 제약점을 가져 설계시간이 길고 고급 프로그래밍 수준이 요구된다
- Backup & Recovery 기법이 어려워진다 ; 데이터 구조가 복잡하며 여러 사용자가 동시에 공유함으로써 장애 발생시 정확한 이유나 상태 파악이 힘들다.
- 시스템의 취약성 ; 통합 시스템이므로 일부의 고장이 시스템 전체를 마비시켜, 신뢰성과 가용성을 저해할 수 있다.

Java 웹

- JAVA플랫폼(J2SE, J2EE)중에서 J2EE를 이용한 웹프로그래밍

  J2EE -> 컨테이너(컴포넌트 관리는 Tomcat이 함) -> 웹 컨테이너안에는 JSP, Servlet, HTML이 담겨 있음. 이 JSP나 Servlet이나 HTML 하나 하나를 컴포넌트라하고 , 이 컴포넌트를 하나하나 완성해 가는 것을 웹 프로그래밍이라 한다.
- 컴포넌트 : JSP, Servlet, HTML 등의 웹어플리케이션을 구현하기 위한 구성요소
- JSP(Java Server Page) ; HTML 파일 내에 java 언어를 삽입한 문서
- Servlet(Server Applet) ; Java 언어로 이루어진 웹 프로그래밍 문서