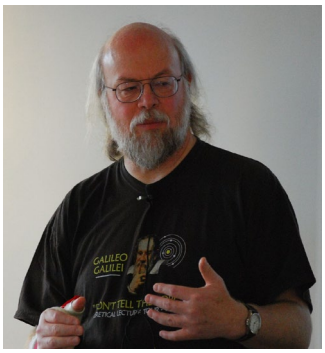
JAVA Programming

목 차



- 제임스고슬링
자바 언어는 1991년 6월 셋톱 프로젝트를 위해 Sun사의
제임스 고슬링의 팀원들과 개발
(C를 기본기초로 만들어 C와 비슷함.)

-JVM(자바 가상 머신) : Java Virtual Machine
※C언어는 한 컴퓨터에서 실행된 것을 다른 컴퓨터에서 실행하지 못한다.
반면에 Java는 JVM을 이용해서 다른 컴퓨터 에서도 이용할 수 있다.
-시스템 동작과정
※ 메모리 - OS operational system - SW

1) JDK와 JRE
-JRE(자바 운영 환경) : Java Runtime Environment
-JDK(자바 개발킷) : Java Development Kit
-> 자바 개발 킷에는 JRE이 포함되어 있으므로 JDK 설치시 JRE는 따로 설치 안해도된다.
2) 설치과정(환경변수 설정)
시스템 변수 설정
-새로만들기 : JAVA_HOME, 주소작성(JDK 저장한 위치)
-Path설정 : path 맨뒤에 " ;%JAVA_HOME%₩bin; "작성
※확인 : cmd창에서 java -version, javac -version(java를 컴파일한 버젼)
◎ 간단한 원리 익히기
① 워드패드에 "파일명.java" 파일 생성
② cmd 창에서 javac + "파일명.java"
- 컴파일하는 과정(폴더를 살펴보면 "파일명.class"가 생긴것을 볼 수있다.)
③ java "파일명" -> 입력된 결과 확인 가능

```
2018-11-13  오후 02:53    <DIR>          .
2018-11-13  오후 02:53    <DIR>          ..
2018-11-13  오후 02:53               453 Hello.class
2018-11-13  오후 02:52               130 Hello.java
               2개 파일                 583 바이트
               2개 디렉터리  74,420,256,768 바이트 남음
```

3) 실행과정

에디터 -> Hello.java

▼

컴파일러-> Hello.class(컴퓨터가 읽을 수 있는 단계로 만드는 과정)

▼

클래스 적재기, 바이트코드검증기 -> 메인메모리

▼

자바가상기계(JVM)

4) 통합개발환경(Integrated development environment)

-> 원래 도스창에 컴파일을 하고 출력하는 일련의 과정들을 복잡하게 처리했는데 이 를 해결하기 위해 통합개발 환경을 개발 하였다. ex) 이클립스, 넷빈 등등등

5) 이클립스 설치

-> JAVA언어 뿐만 아니라 다른 언어들도 개발 가능.(C, C++, JSP 등)

-> JAVA만 개발하기에는 Eclipse IDE for JAVA Developers가 좋다.

-> JAVA 이외에 웹개발, 배포를 위해 Eclipse IDE for Java EE Developers를 사용.

① 이클립스 실행

코드작성 -> 클래스 파일 밑에 src폴더(파일명.java 생성)

-> workspace 하위폴더(bin) : 컴파일 된 class파일 생성

② 단축키

-Ctrl + F11 : 코드 실행

-Ctrl+Shift+f(범위 지정 후) : 코드 정리 단축키

2.Java 기초

1.주석:

- // 문자(문장주석)
- /* 문자 */(문단주석)
- /** 문자 */(문서주석) : 문서전체를 설명할때 사용

-> 문단주석 단축키(범위 지정후) ctrl+ shift +c ctrl+/

2.용어정리

- 클래스 : 객체를 설계하는 기본단위
- 함수 : 값을 입력하면 어떤 과정을 통해 결과를 내놓는 것 x-> F(x) (x의 값의 따라) -> y
- 변수 : 변하는 값, 데이터를 메모리에 저장하는 공간

  ex) 컵에 어떤 물체를 담느냐에 따라 컵의 용도가 달라진다.

  ※변수는 타입과 이름을 갖는다

- 상수 : 변하지 않는 값
- 초기화 : 변수를 선언한후 변수에 선언한 타입한 값을 넣는것 int x;(선언) -> x=100;(초기화)
- ※선언을 하는 순서에 따라 메모리에 저장이 된다.
- 대입연산자 == 할당연산자
- System.out(표준출력) vs System.in(표준입력)

① System.out(표준출력) : 모니터로 데이터를 출력할 수 있는 기능

② System.in(표준입력) : 키보드로 부터 데이터를 입력받을 수 있는기능

- import java.util.Scanner;

  컴파일러 한테 java안에 util에 Scanner라고 있으니 준비해놔!!! 라는 명령

- Scanner input = new Scanner(System.in);
- 타입 변수명 = 새로운 스캐너에("표준입력")을 받는다.

```
import java.util.Scanner; // Scanner를 쓸거니까 Compiler야 Scanner 불러와놔
public class Add2 {
 public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   int x, y, sum; //변수 x, y, sum 선언

   System.out.println("첫번째 숫자를 입력하시오 : ");
   x = input.nextInt(); // 키보드 입력 들어오는 다음의 값을 정수로 읽어서 갖다줌,
                        그리고 그 값을 x에저장
   System.out.println("두번째 숫자를 입력하시오 : ");
   y = input.nextInt();
   sum = x + y;
   System.out.println("합은 :" + sum + "입니다");
 }
}
```

3. JVM 메모리구조

int x;, in y;, int sum; Scanner input =new Scanner(System.in); // 이렇게 만들었다고 했을때
-> 변수가 선언되는 것은 Class area 부분에 저장이되고 Heap에는 Scanner의 객체가 생성이 된다.
그리고 Stack에서는 Heap에서 생성된 Scanner객체에 접근할 수 있는 주소값이 만들어진다.

4. 오류

1) 컴파일 오류(compile-error) : 컴파일 실패(예를 들어서 문법 틀려서)
-> 대놓고 에러 뱉어냄.. 발견가능
2) 실행오류(run-time error) : 실행 도중에 프로그램이 뻗어버림 ex) 블루스크린, 값을 잘못 입력했거나
-> 충분한 시간을 갖고 테스트를 하면 답이 나옴
3) 논리오류(logical error) : 컴파일/ 런타임 에러없이 잘작동됨.. 프로그램의 결과 값이 내가 원하는 값이 아님..!!

5. 변수

1) 변수의 이름은 식별자의 일종
2) 변수이름의 규칙
① 식별자는 유니코드 문자와 숫자의 조합(x1, x2, n1.....)
② 식별자의 첫문자는 일반적으로 유니코드 문자(알파벳..관례적으로)
③ 두번째 문자부터부터는 문자, 숫자,_,$ 등이 가능하다.
④ 대문자와 소문자는 구별된다.(첫글자는 소문자, 연결된 두번째 단어는 대문자)
⑤ 식별자의 이름으로 키워드를 사용해서는 안된다.(코드내 보라돌이들 public, import 등)

3. 선택과 반복

1.If 문(If-else문)

사실 if-else문이라는 것은 없다 if문이 맞는 말이다. if문은 조건문의 결과가 참이라고 하면 그때 if문을 실행을 한다. else는 짝꿍(가장 가까운) if문의 조건식이 False일때 실행이 된다.

```
public class Add2 {

 public static void main(String[] args) {
  int score = 93;

  if (score <= 90)
   System.out.println("점수가 90점보다 큽니다.");
   System.out.println("냠냠.");
  }
} //중괄호 없이 if문을 수행했을 경우에 if문은 자기 아래의 구문까지만 인식
```

```
import java.util.Scanner;
/**
 * if-else 문을 사용하여 입력된 성적 분류하기
 */
public class IfTest {
  public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   int score;
   System.out.println("과제의 점수를 입력하세요 : ");
   score = input.nextInt();
   if (score >= 90) {
    System.out.println("A");
   } else if (score >= 80) {
    System.out.println("B");
   } else if (score >= 70) {
    System.out.println("C");
   } else if (score >= 60) {
    System.out.println("D");
   } else {
    System.out.println("바보");
   }
  }
}
```

3. 선택과 반복

■Math.random(); <- 0.000.... ~ 0.9999...까지의 임의에 난수를 생성함.
정수로 가공해서 만들고 싶다. Math.random() 함수앞에 (int)를 붙여
강제형변환을 시켜준다. ex) (int)Math.random()*6 --> 0 ~ 5

-> *7을 해서 1~6까지 표현할 수 있을 거라고 생각할 수도 있지만 어떤 값을 곱해도 앞에 정수 값은 0이 나오기 때문에 코딩을 할때에는 +1을 하여 최초의 값을 1로 표현을 해주어야 한다.

(int)(Math.random()*6)+1 -> 1 ~ 6 (코딩을 할때 1부터 6까지 표현을 할 수 있다)

-> 6이라는 값은 정해져 있는 값이 아니고 조건에 따라 바뀔수있다.

2. switch-case문

1) 형식 :

```
switch (number) {  // switch 문은 if문과 다르게 조건식이 아닌 자료형을 넣음
  case 0: // 그리고 위의 조건이 OK할 경우에 아래의 case의 번호에 맞는 곳에 들어감
   break; // 원하는 값만 출력하고자 할 때에는 꼭 break를 넣어주어야 함
  case 0:
   break;
  default:// 모두 부합하는 조건이 없다면 default 값을 출력
   break;
}
```

3. while

1) 형식

while(조건식)

{ 문장; }

2) 원리 이해

- 반복계수 : 반복문 수행 전 초기화 되고 조건에 영향을 주고 반복구문 안에서 증감되는 변수

```
int i = 1; // 1
while (i < 2) { // 2
 System.out.println("정수:" + i); // 3
 i++; // 4
}
```

-코드가 돌아가는 순서

i=1; (메모리 저장) -> while(조건) ok!! -> 출력 -> 증감(i=2)(메모리 저장)

->while(조건) False!! ->종료

* 마지막에 증감을 하고 나올 것 같지만 조건문을 확인을 하고 나오게 된다.

4. for

1) 형식

for(초기화; 조건식; 증감식)

{ 반복할 문장; }

Ex)

```
for (int num = 1; num < 4; num++)
{
  System.out.println(num);
}
```

2) 원리 이해

- 1번 int num=3(메모리 저장) -> 2번 조건값(True) -> 4번 (출력) ->
3번 num = 4(증감)(메모리 저장)-> 2번 조건 값(False)->끝

4. 배열

1 . 배열 : 같은 타입의 변수들의 모임

1) 변수 선언

① 선언

1) int[] array = new int [선언할 배열의 수(array의 index)]

변수 타입 = * array는 참조변수(주소값을 담고 있는 변수)

* 힙 영역에 연속된 변수의 set이들어감, 배열의 시작은 무조건 0번 부터 시작이 됨.

2) int[] array = { a,b,c,d }

2) 동적 vs 정적

-동적 : 프로그램이 실행되는 중에 생성되는 것

-정적 : 컴파일을 하면서 부터 생성이 되있는것

3)final : 상수를 나타내며 final로 선언을 했을 때 값은 변하지 않는다. , 배열.length : 선언한 배열을 끝까지 읽어 올때 많이 쓴다.

4)for -each : 배열전체를 읽어올때 사용

for(int value : numbers)

해당 배열과 같은 타입의 변수, 뽑아올 대상 배열

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

   int[] arr = { 5, 7, 8, 2, 6 };
   int value1; // 값
   int pos; // 위치 표시

   for (int j = 0; j < arr.length; j++) { //배열의 길이까지 반복을 실행함
     value1 = arr[j]; // 배열의 값을 저장
     pos = j; // 변경된 위치를 표시함
   for (int i = j; i < arr.length; i++) { //순차적으로 배열의 반복을 실행하며
// 변경된 위치를 초기값으로 잡고
        if (value1 > arr[i]) { // 기준점 이후를 비교하여 값이 크면
          value1 = arr[i]; // 그 값을 value 값에 다가 저장을 한다 .
          pos = i; // 위치를 변경한다.
        }
    }
    int tmp = arr[pos]; // 변경할 위치의 값을 임시 변수에 저장
    arr[pos] = arr[j]; // 변경할 값을 대입
    arr[j] = tmp; // 변경 완료
   }
   for (int x : arr) // for-each 문을 사용해 배열 전체를 읽어옴
     System.out.print(x);
   }
}
```

1. 구조체 : 서로 다른 타입의 변수들의 묶음 자료형 -> 사용자 정의 자료형
*기초자료형 : int ,double, char 등
-> 사실 구조체라는 말은 C에서 쓰는 말이고 JAVA에서는 쓰지는 않는다.
이해를 위해 구조체를 표현함.

1-1. 선언방법

* 참조변수 : heap에 만든 객체에 접근하기 위해 stack에 저장된 주소 값

[Student를 Heap영역에 저장하기 위해서 new라는 연산자를 사용해서 만듦
new 연산자는 class student를 보고 heap영역에 할당함(객체화)]

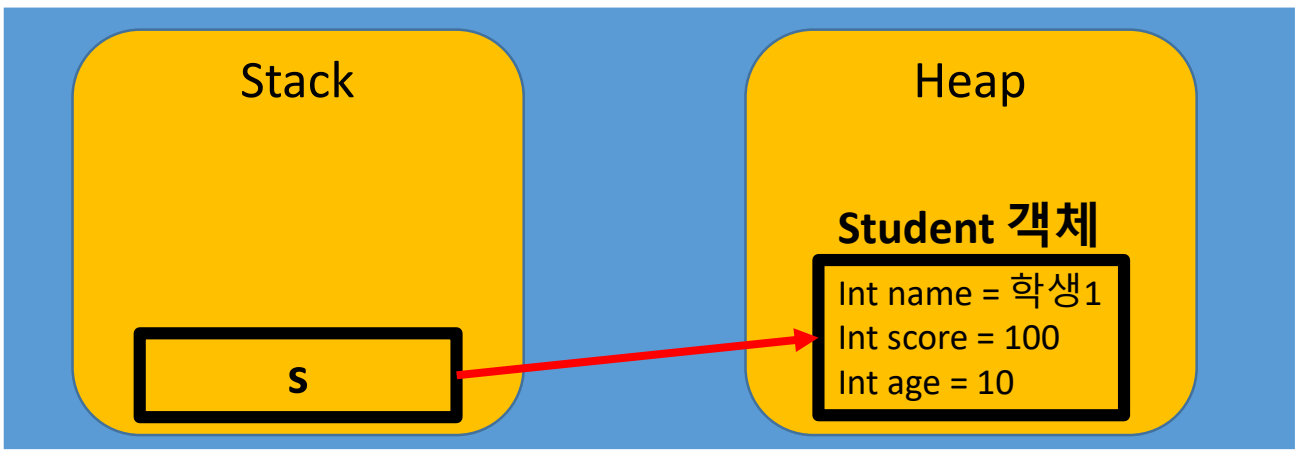
```
class Student {
   int age;
   int score;
   String name;
} // 구조체 선언 : 다른타입의 변수들의 묶음 자료형[사용자 정의 자료형]

public class Test {

   public static void main(String[] args) {

      Student s = new Student();
      // Student라는 구조체의 s 참조변수를 하여 Heap에 있는 객체에 접근하기 위함.
      s.age=10;
      s.score = 100;
      s.name = "학생1";

      System.out.println("나이:" + s.age);
      System.out.println("나이:" + s.score);
      System.out.println("나이:" + s.name);
   }
}
```



서로 다른 타입의 변수들을 선언하고 참조변수를 통해 접근 가능.

ex) s.age; s라는 참조변수를 사용하여 Heap에 있는 Student 객체의 age에 접근

2. 함수(메소드)

1) 정의 : **명령어들의 집합**

프로그램을 작성 하다보면 특정 명령어 집합이 반복됨. 그때마다 코드를 복사, 붙여넣기 하기에는 무리가 있음. 나중에 다시 사용될 가능성이 높은 코드들을 모아놓고 그 집합에 이름을 지어주면 끝.

2) 인자값, 매개변수

① 인자값 : 함수를 호출하기 위해서 호출시에 전달하는 값

② 매개변수 : 전달된 값을 받기 위한 변수[인자값을 받는 변수]

printStudent(Student stu) -> 아래와 코드를 작성한 것과 같이 함수(클래스 이름 타입) 식으로 작성을 한다.

```
class Student {
    int age;
    int score;
    String name;
}

public class Test {
    public static void printStudent(Student stu) { //Student 타입의 매개변수 stu
        System.out.println("나이 :" + stu.age);
        System.out.println("점수:" + stu.score);
        System.out.println("이름:" + stu.name);
    }

    public static void main(String[] args) {

        Student s = new Student();
        s.age = 10;
        s.score = 100;
        s.name = "학생1";

        printStudent(s); // 인자값
    }
}
```

5. 객체지향소개

1. 객체지향 ?

실제 세계를 모델링하여 소프트웨어를 개발하는 방법

객체 : 관련있는 변수와 함수를 하나를 꾸러미로 묶어서관리 *객체는 상태와 동작을 갖는다.

1) 절차 지향과 객체지향

① 절차지향 : 데이터와 알고리즘이 묶여 있지 않다.

② 객체지향 : 데이터와 알고리즘이 묶여있다.

2) 상태와 동작

① 상태 : 변수(특징값, 속성) -> 변수들의 집합 -> 클래스에서 필드에 해당

② 동작 : 함수(객체가 취할 수 있는 동작) -> 함수(메소드) -> 클래스에서 메소드에 해당

자바에서의 객체는 Heap 영역에 할당되어 있는 모든데이터

2. 클래스 : 객체를 만드는 설계도 * 클래스 -> 인스턴스(new) -> 객체

■용어정리

1_클래스 : 관련있는 변수와 관련있는 함수를 조합해서 만든 자료형(data type)

2_인스턴스 : 클래스를 사용해 인스턴스화시켜(객체를 만드는 과정) 생성해낸 데이터를 말함

Idea..!! 건물을 지을 때 설계도면을 가지고 건물 기초 부터 쌓아 올라간다. 한 층씩 쌓아가며 최종적으로 건물이 완성하게 된다. 이 과정에서 설계도면이 클래스이고 건물을 완성하기 까지의 과정이 인스턴스화이고 최종적으로 건물이 완성된것이 객체를 의미한다.

3_객체 : 클래스를 통해 생성해낸 데이터

```java
public class CarTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Car abbaCha = new Car();
        Car ummaCha = new Car();
        abbaCha.speed = 100;
        abbaCha.color = "Pink";
        abbaCha.mileage = 1000;
        ummaCha.speed = 200;
        ummaCha.color = "Read";
        ummaCha.mileage = 2000;

        abbaCha.print();
        ummaCha.print();
        abbaCha.speeddown(30);
        ummaCha.speedup(50);
        abbaCha.print();
        ummaCha.print();

        System.out.println(abbaCha.toString());
    }
}
```

```java
class Car {
    // 상태 : 속도, 색상, 주행거리
    int speed;
    String color;
    int mileage;
    // 기능 : 가속, 감속, 현재상태 출력
    public void speedup(int s) {
        speed += s;
    }
    public void speeddown(int s) {
        speed -= s;
    }
    public void print() {
        System.out.println("speed 의 상태는 : " + speed);
        System.out.println("Color 의 상태는 : " + color);
         System.out.println("mileage 의 상태는 : " + mileage);
    }
    public String toString() {
        return
        "현재속도 : " + speed + "\n색상 : " + color + "\n주행거리 : " + mileage;
    }
}
```

5-2 객체지향소개 [접근제어, getter&setter, 생성자, 정적변수]

1. 접근 제어 : 다른 클래스가 특정한 필드나 메소드에 접근하는 것을 제어하는 것

1) private - 전용멤버 : 클래스 안에서만 사용

ex) 내가 소유한 재산, 집 등으로 생각할 수 있음

2) package - 패키지 멤버 : 같은 패키지 안에서만 사용

3) public - 공용멤버 : 모든클래스가 사용 가능

ex) 사회로 따지면 공공 시설

■클래스의 멤버변수에 private 설정

```
public class Car {
    private int speed;
    private String color;
    private int mileage;
}
```

2. 설정자(setter), 접근자(getter)

필드에 설정된 변수의 값이 Private으로 설정이 되어있을 때 사용.

1) 용어정리

① 설정자 : 필드의 값을 설정하는 메소드 setter

② 접근자 : 필드의 값을 반환하는 메소드 getter

2) 사용이유

① 외부에서 남의 데이터를 직접 접근하는 것을 막기위해

② 세분화된 접근제어

③ 입력값에 대한 검증이 가능

3) 분류

① 접근자(getter) 만 있으면 읽기 전용

② 설정자(setter) 만 있으면 쓰기 전용

③ 접근자(getter), 설정자(setter) 가 모두 있으면 읽기, 쓰기 전용

④ 다 없으면 접근 불가

4) 설정자(setter), 접근자(getter) 만들기

① 설정자(setter) 만들기

1_수식어는 public

2_반환유형 void

3_메소드 명은 set+ 변수명(변수명은 첫글자를 대문자로) ex) setMethod..등

4_매개변수 : 내가 set 하려고 하는 변수의 타임

5_몸통 : 내가 setter가 되고자 하는 변수(멤버변수) = 매개변수;

② 접근자(getter) 만들기

1_수식어는 public

2_반환유형은 내가 get 하고자 하는 변수의 타임

3_메소드명 get + 변수명(변수명은 첫글자를 대문자로 ex) getSpedd; getColor, getMileage 등..

4_매개변수는 필요하지 않다.

5) 설정자(setter), 접근자(getter) 쉽게 만들기

클래스를 생성하면 멤버를 생성을하고 getter와 setter를 만들어야한다.

① 단축키 : Alt + Shift + s

② source-> generate getter, setter

3. 생성자(contructor)

객체가 생성될 때에 필드에게 초기값을 제공하고 필요한 초기화 절차를 실행하는 메소드

* 생성자 : 객체가 생성될때(new) 무조건 한번은 수행되는 메소드

1) 생성자의 특징

① 수식어는 웬만하면 public

② 문법적으로 반환유형 자체가 없음

③ 얘도 메소드이니까 오버로딩이 가능

④ 생성자 메소드의 이름은 클래스명과 완전동일

-> 2,4 번 두가지를 모두 만족해야 생성자

■메소드 오버로딩

함수 이름이 같은데 매개변수가 다른 두개 이상의 함수를 정의하는 것

```
// 오버로딩의 예시
public void speed() {//1번
}
public void speed(int x) {//2번
}
public void speed(double y) {//3번
}
public void speed(int x, int y) {//4번
}
```

생성자를 호출할때 매개변수가 아무것도 없을 때에는 1번을 호출,

파라미터가 정수타입 한개일 경우에는 2번,

파라미터가 실수타입 한 개 일 경우에는 3번,

정수타입 2개의 파라미터를 갖고 있을 경우에는 4번이 호출이 된다.

2) 기본생성자, 사용자 정의형 생성자

1_기본생성자 : 매개 변수가 없는 생성자 -> ex) Car ummaCha = new Car();

2_사용자 정의형 생성자 : 매개변수가 있는 생성자 -> ex) Car abbaCha = new Car(1,"gu",3);

■생성자 특징

① 생성자를 하나도 정의하지 않았을 경우에는 몸통이 텅 비어있는 {} 기본 생성자를 컴파일러가 잠시 끼어넣음

② 사용자 정의형 생성자만 있을 경우에는 The constructor Car(int, String, int) is undefined 라는 에러 발생

③ 사용자 정의형 생성자와 기본생성자가 있는데 사용자 정의형 생성자의 매개변수만 받는 함수가 있을경우 기본생성자가 자동생성해주는 서비스를 해주지않음. 즉, 기본 생성자 부분에서 에러발생

-> 컴파일 입장에서는 두번 일할 필요가 없기 때문. 만약 기본생성자를 쓰고 싶으면?

-> 기본생성자를 한 개 만들어주면 됨

```
// 생성자 특징 3번 확인 코드
class Car {
    public int speed; // 속도
    public int mileage;// 주행거리
    public String color; // 색상

    public Car(int s, String c, int m) { // 사용자 정의형 생성자의 매개변수를 받는 함수
        System.out.println();
    }
}

public class CarTest {
    public static void main(String[] args) {
        Car ummaCha = new Car(); // 기본생성자-> 에러발생!
        Car abbaCha = new Car(1, "gu", 3); // 사용자 정의형 생성자 인자값 입력
    }
}
```

4. 정적변수

■정적변수 들어가기전에..

멤버변수, 멤버함수 : 클래스를 구성하는 요소로 클래스를 통해 객체를 생성하면 각 객체마다 새로운 멤버변수, 멤버함수를 생성해냄

1) 정적변수 : 모든 객체를 통틀어서 하나만 있는 변수

멤버변수와는 다르게 객체를 생성하지 않아도 아무리 많이 생성해도 딱 한개만 생성되는 변수

-> static 키워드를 쓰면 딱한개만 만들어줌

2) 인스턴스 변수 : 객체마다 하나씩 있는 변수

3) 정적변수 사용

정적변수는 [객체참조변수.정적변수] 둘 다 접근 가능하지만... 기본적으로 정적변수는 [클래스명.정적변수]로 접근하는게 더 맞다.

왜냐하면 객체가 없을 때도 변수는 존재하고 또 그때도 정적변수를 사용 하려면 당연함!

4) 정적변수 용도

모든객체가 하나의 데이터를 공유하기 위해서 사용

■변수 정리

① 지역변수 : 메소드 내부에서 생성, 스택(main 함수 내에서 int x 등등) 메소드가 종료될 때 소멸

② 멤버변수 : 클래스 내부에서 정의, 객체가 생성될 때 힙에 생성, 객체가 소멸될 때 같이 소멸

③ 정적변수 : 클래스 내부에 static키워드로 정의, 프로그램 실행될 때 생성, 프로그램 종료될 때 소멸

6. 상속

1. 상속 - 이어붙인다

1) 개념

1_ 어떤 클래스가 다른 클래스의 멤버변수와 멤버함수를 물려 받는것

2_ 상속은 코드를 재사용하기 위해서 사용함

2) 장점

1_상속을 통하여 기존 클래스의 필드와 메소드를 재사용

2_ 기존 클래스의 일부 변경도 가능

3_ 이미 작성된 검증된 소프트웨어를 재사용

4_신뢰성 있는 소프트웨어를 손쉽게 개발, 유지보수

5_ 코드의 중복을 줄일 수 있음.

3) 상속하는 방법 : extends 사용

```
class Car {
    int speed;
}

class SportsCar extends Car {
    int tubo;
}
```

슈퍼클래스 == 부모클래스 == 베이스클래스(Car)
서브클래스 == 자식클래스 == 파생된클래스(SportsCar)

2. 접근 지정자

1)private : 부모가 private를 설정해 놓으면 상속받은 자식의 영역에서 접근이 불가함.

2)protected : 같은 패키지와 자신을 상속받은 자식 클래스 영역에서 접근가능

■코드이해

6. 상속

```
class Parent {
    int data = 100;
    public Parent() {
        System.out.println("Parent의 기본생성자");
    }
    public void print() {
        System.out.println("부모임");
    }
}

class Child extends Parent {
    int data = 200;

    public Child() {
        super(); // 이 코드는 생략되어도 됨. // 묵시적 호출(생략이 되어도 자동적으로 호출함)<-> 명시적 호출
        System.out.println("Child의 기본생성자");
    }

    public void print() {
        int data = 300;
        super.print();
        System.out.println("자식임");
        System.out.println("data :" + data);
        System.out.println("this.data : " + this.data);
        System.out.println("super.data : " + super.data);
    }
}

public class ParentTest {
    public static void main(String[] args) {
        new Child().print();
    }
}
```

출력결과 :

```
Parent의 기본생성자
Child의 기본생성자
부모임
자식임
data :300
this.data : 200
super.data : 100
```

3. Object클래스

1) Objec클래스는 상속계층에서 맨위에 해당

2) Object의 메서드

1_ protected Object clone() : 객체 자신의 복사본을 생성하여 반환한다

2_ public boolean equals(Object obj) : 이 객체와 같은지 비교

3_ public final Class getClass() : 객체의 실행 클래스를 반환한다

4_ public int hashcode() : 객체에 대한 해쉬코드 반환한다

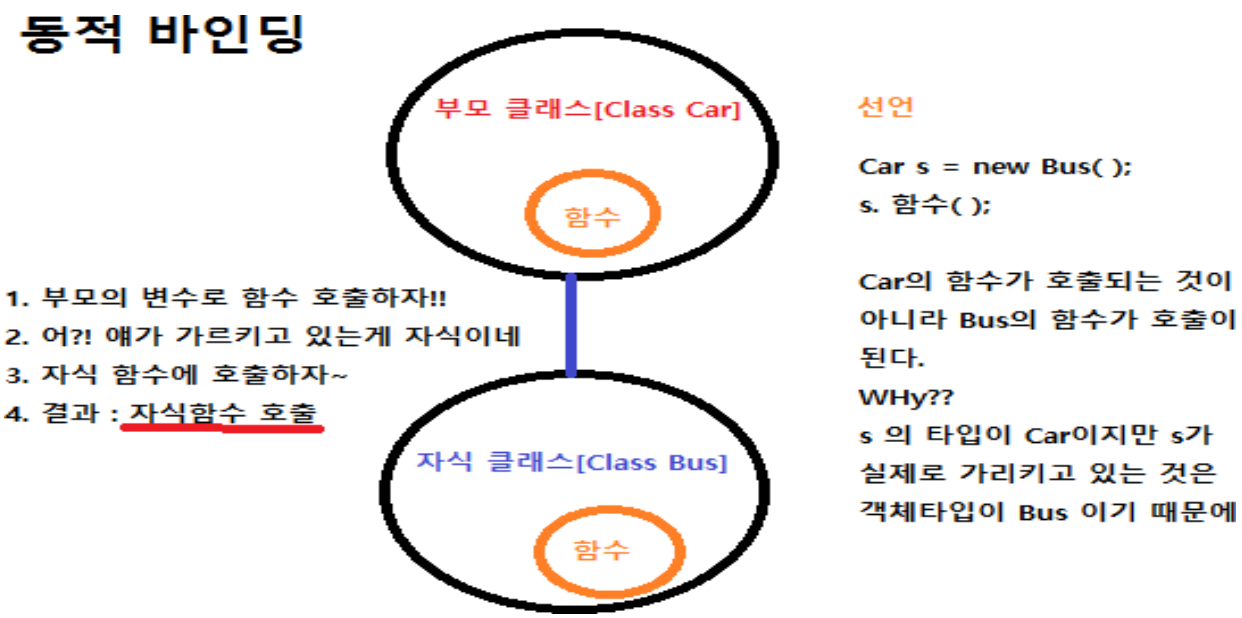
5_ public String toString() : 객체의 문자열 표현을 반환한다

4. 객체지향에서의 다형성

1) 전제 : 부모클래스의 참조변수로 자식클래스의 객체를 참조할 수 있음.

-> 자식클래스의 객체에는 부모클래스로 만든 객체를 포함하니까 당연함.

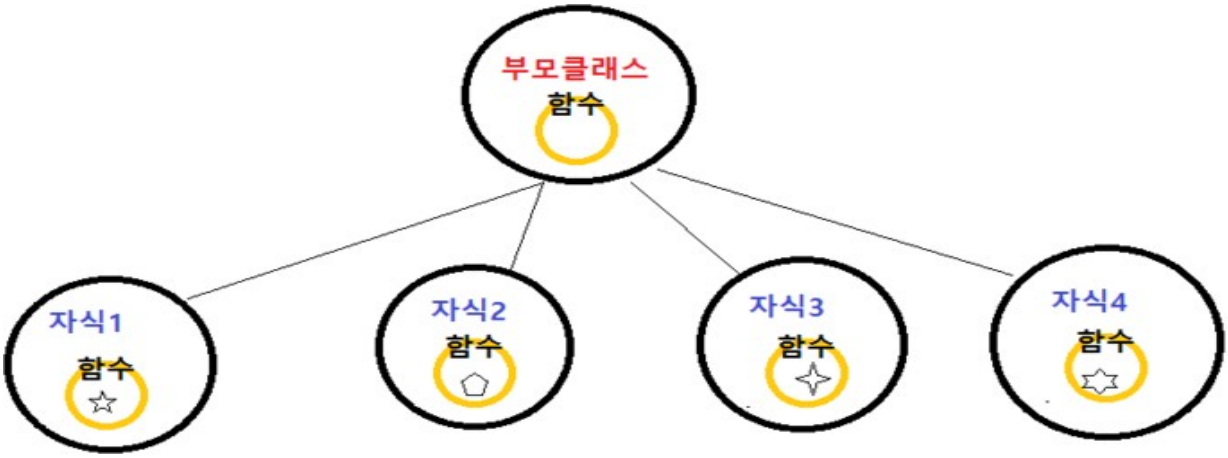
2) 부모클래스의 참조 변수로 자식 클래스 객체를 참조했을 때는 부모클래스에 존재하는 멤버에만 접근이 가능함.



6. 상속

■오버라이딩(overriding)

자식클래스가 자신의 부모클래스의 메소드를 특정한 형태로 구현하는 것



오버라이딩은 부모클래스의 메소드를 가져와서 자신에게 맞는 형태로 구현하는 것을 의미한다.

■형변환

부모클래스 타입으로 자식 객체를 참조했을 때는 부모클래스에 존재하는 멤버변수와 멤버함수에만 접근 가능하다. 자식영역에만 존재하는 멤버에는 접근이 불가능하다.

7. 추상클래스, 인터페이스

1.추상클래스

몸체가 구현되지 않은 메소드를 가지고 있는 클래스, 특정기능이 어떻게 구현될지 애매모호 한 애들

1) 사용방법

추상메소드는 abastract 키워드를 달아줌.

■추상클래스 특징

1. 추상메소드를 하나라도 가지고 있으면 추상클래스이여야 한다.
2. 추상클래스는 미완성의 설계도이기 때문에 객체를 만들 수 없음(객체화 할 수 없음)

■추상메소드 상속받는 방법

1. 추상메소드를 구현하든가.
2. 자기 자신도 추상클래스가 되든가 둘중하나 선택해야 한다.

2. 인터페이스

추상 메소드들로만 이루어진 클래스

■사용시기

인터페이스를 사용하는 경우, 다중상속이나 처음에 설계할 경우에 사용한다.

■인터페이스 특징

1_인터페이스도 클래스처럼 하나의 타입이다.

2_인터페이스 타입의 참조변수는 자신 인터페이스를 implements 한 클래스의 객체를 참조 할 수 있다.(상속과 동일 개념) 대신에 인터페이스 타입의 참조변수로 객체를 참조했을 때 는 인터페이스에 존재하는 메소드만 호출이 가능

■템플릿 메소드 패턴

부모클래스에서 템플릿에 해당하는 기능이 정의 돼있고 메소드내용중 세부적인 기능에 대한 정의는 추상메소드로 남겨둔채 자식클래 스에 세부적인 기능을 구현해서 메소드 탬플릿을 완성하는 방법

-> 즉, 부모클래스에 공통으로 사용될 메소드의 틀을 만들고 그 안에 들어가는 세부적인 기능은 자식 클래스에 구현에 메소드를 쓸수 있 게 만드는 것

-> 순서 1) 부모클래스 : 메소드 템플릿 만들기 2) 세부 기능 다른 부분 추상메소드 선언(abstract) 3) 자식클래스에 세부 기능 만듦

■템플릿 메소드의 단점

1. 코드가 짧을 때는 가능하지만 코드가 길어지고 자식클래스가 많아질 수록 일일이 입력해주어야 하는 단점이 있다.

-> 그래서 반복되는 기능에 대해서는 묶어서 만들수 있음. -> 인터페이스 사용(기능들의 특성에 대 해서 묶어서 사용할 수 있다.)

8. 패키지

1. 패키지 : 연관된 클래스들을 묶는 기법

- 자바라이브러리도 패키지로 구성되어 있음

2. 패키지의 장점

1) 관련된 클래스들을 쉽게 파악

2) 원하는 클래스들을 쉽게 찾을 수 있다.

3) 패키지마다 이름공간을 따로 각지 때문에 같은 클래스 이름을 여러 패키지가 사용

4) 패키지별로 접근에 제약을 가할 수 있다.

3. 디폴트 패키지(default package)

디폴트 패키지 혹은 현재 작업하는 소스파일과 같은 같은 패키지가 아닌곳에 위치한 클래스를 사용하려면 풀패키지명으로 접근 혹은 import를 해야됨

4. 자바 지원 클래스 메소드

1) Math 클래스 메소드

static double max(double a, double b)	큰수
static double min(double a, double b)	작은 수
static double pow(double a, double b)	a^b
static double random()	0.0과 1.0사이의 난수를 반환
static double sin(double a)	sine
static double sinh(double x)	hyperbolic sine
static double sqrt(double a)	제곱근
static double tan(double a)	tangent
static double tanh(double x)	hyperbolic tangent
static double toDegrees(double angrad)	라디안을 디그리로 변환
static double toRadians(double angdeg)	디그리를 라디안으로 변환

필드	설명
static double E	자연 로그의 밑수(the base of the natural logarithms.)
static double PI	파이값

메소드	설명
static double abs(double a)	절대값
static double acos(double a)	arc cosine, 반환값의 범위는 0.0에서 *pi*.
static double asin(double a)	arc sine, 반환값의 범위는 -*pi*/2에서 *pi*/2.
static double atan(double a)	arc tangent, 반환값의 범위는 -*pi*/2에서 *pi*/2.
static double atan2(double y, double x)	직교 좌표계 (x, y)를 극좌표계 (r, *theta*)로 변환할 때 theta를 반환
static double cos(double a)	cosine
static double cosh(double x)	hyperbolic cosine
static double exp(double x)	e^x
static double hypot(double x, double y)	sqrt(x^2 +y^2)
static double log(double a)	natural logarithm (base *e*)
static double log10(double a)	base 10 logarithm

2) Wrapper 클래스 : 기초 자료형을 객체로 포장시켜주는 클래스

기초자료형을 객체로 바꿔주면 int ->Integer, char-> Character, double -> Double 등 바뀜

```
public class Test {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("숫자 1입력 : ");
        String n1 = input.nextLine();
        System.out.println("숫자 2입력 : ");
        String n2 = input.nextLine();
//        int num1 = n1;
//        int num2 = n2;
//       위에 처럼 쓰면
        int num1 = Integer.parseInt(n1);
        int num2 = Integer.parseInt(n2);
//        Wrapper클래스
//        parseInt 문자형으로들어왔을 때 바꿀수 있는 건 바꾸고 안되는 것은 안된다.
        int result = num1 + num2;
        System.out.println(result);
    }
}
```

3) Integer클래스 메소드

반환값	메소드 이름	설명
static int	intValue()	int형으로 반환한다.
static double	doubleValue()	double형으로 반환한다.
static float	floatValue()	float형으로 반환한다.
static int	parseInt(String s)	문자열을 int형으로 변환한다.
static String	toBinaryString(int i)	int형의 정수를 2진수 형태의 문자열로 변환한다.
static String	toHexString(int i)	int형의 정수를 16진수 형태의 문자열로 변환한다.
static String	toOctalString(int i)	int형의 정수를 8진수 형태의 문자열로 변환한다.
static String	toString(int i)	int형의 정수를 10진수 형태의 문자열로 변환한다.
static Integer	valueOf(String s)	문자열 s를 Integer 객체로 변환한다.
static Integer	valueOf(String s, in radix)	문자열 s를 radix진법의 Integer 객체로 변환한다.

4) 오토박싱(auto-boxing)

Wrapper 객체와 기초 자료형 사이의 변환을 자동으로 수행한다.

Integer box = new Integer(10);

System.out.println(box+1); // box는 자동으로 int 형으로 바뀌어서 연산이 가능하다.

5) StringBuffer 클래스 메소드

String 클래스는 주로 상수 문자열, 즉 변경이 불가능한 문자열을 나타낸다.

-> String s ="Hello";라고 선언을 한다고 하면 이것은 class 영역에 저장이된다.

Why? String도 객체이기 때문이다

```
String s1 = "Hello";
String s2 = "Hello";
String s3 = new String("Hello");
String s4 = new String("Hello");

System.out.println(s1 == s2); // True
System.out.println(s3 == s4); // False s3, s4는 주소값을 가르키고 있기때문에 다름
```

메소드	설명
StringBuilder append(String s) StringBuilder append(char[] str) StringBuilder append(char[] str, int offset, int len) ...	인수는 먼저 문자열로 변환되어서 문자열에 붙여진다.
StringBuilder delete(int start, int end) StringBuilder deleteCharAt(int index)	지정된 문자를 삭제한다.
StringBuilder insert(int offset, char[] str) StringBuilder insert(int index, char[] str, int offset, int len) ...	첫번째 매개 변수는 데이터가 삽입될 위치의 바로 앞을 나타낸다. 두번째 매개 변수는 문자열로 변환된 후에 삽입된다.
StringBuilder replace(int start, int end, String s) void setCharAt(int index, char c)	지정된 위치의 문자를 변경한다.
StringBuilder reverse()	저장된 문자들의 순서를 역순으로 한다.

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

        StringBuffer sb = new StringBuffer();
        // 1번 방법
        String str = sb.append("Hello ").append("world ").append("\n").append("how are you? : ").toString();

        // 메소드의 반환값이 자기 자식객체임. 그래서 메소드 호출 결과에 다시한번 메소드를 호출..을 반복함 메소드체인기법 빌더패턴
        // 2번 방법
        sb.append("Hello ");
        sb.append("world");
        sb.append("\n");
        sb.append("welcome ");
        sb.append("how are you? : ");
        String str2 = sb.toString();
        System.out.println(str);
        System.out.println(str2);

    }
}
```

6) Date 클래스

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

public class DateTest {

    public static void main(String[] args) {

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-mm-dd:hh.mm.ss"); // 시간을 다음과 같은 포맷에 만들어
        String str = sdf.format(new Date()); // 그시간을 포맷 시켜서 str에 집어 넣어라 라는 말임.
        System.out.println(str);

        Date d = new Date();
        System.out.println(d.getTime());

        Date dd = new Date(System.currentTimeMillis());
        System.out.println(dd);

        System.out.println(d);
        System.out.println(d.getYear() + 1900); // 연도
        System.out.println(d.getMonth() + 1); // 월
        System.out.println(d.getDate()); // 일
        d.setHours(12);
        d.setMinutes(00);
        d.setSeconds(38);
    }
}
```

1) Calendar클래스 메소드

Calendar는 객체임에도 불구하고 생성자를 통해 만들수가 없다. 왜냐하면 Calendar는 private로 설정되어있기 때문이다. 그래서 getInstnance를 이용해 객체를 만들어줄 수 있다.

메소드	설명
abstract void add(int field, int amount)	지정된 필드에 시간을 더하거나 뺀다.
boolean after(Object when)	현재의 객체가 주어진 시간보다 뒤이면 true를 반환한다.
boolean before(Object when)	현재의 객체가 주어진 시간보다 앞이면 true를 반환한다. .
void clear(int field)	지정된 필드를 정의되지 않은 상태로 변경한다.
int compareTo(Calendar anotherCalendar)	두개의 Calendar 객체를 비교한다.
boolean equals(Object obj)	두개의 Calendar 객체가 같으면 true를 반환한다.
int get(int field)	주어진 필드의 값을 반환한다.
static Calendar getInstance()	현재 시각을 나타내는 Calendar 객체를 반환한다.
Date getTime()	Calendar 객체를 Date 객체로 반환한다.
void set(int year, int month, int date, int hourOfDay, int minute, int second)	각 필드의 값을 설정한다.
void setTime(Date date)	Date 객체의 값으로 Calendar 객체를 설정한다.

2) StringTokenizer 클래스 메소드

문자열을 분석하여서 토큰으로 분리시켜 주는 기능을 제공

생성자
StringTokenizer(String str) 주어진 문자열을 위한 StringTokenizer 객체를 생성한다.
StringTokenizer(String str, String delim) 주어진 문자열을 위한 StringTokenizer 객체를 생성한다. 분리자로 delim을 사용한다.
StringTokenizer(String str, String delim, boolean returnDelims) 주어진 문자열을 위한 StringTokenizer 객체를 생성한다. 분리자로 delim을 사용한다. returnDelims이 true이면 분리자를 포함하여 반환한다.

메소드	설명
int **countTokens**()	문자열에 존재하는 토큰의 개수를 반환한다.
boolean **hasMoreTokens**()	다음 토큰을 가지는지를 반환한다.
String **nextToken**()	다음 토큰을 반환한다.
String **nextToken**(String delim)	다음 토큰을 반환하고 분리자를 delim으로 변경

```
import java.util.StringTokenizer;

public class Test {

    public static void main(String[] args) {

        StringTokenizer st = new StringTokenizer("will Java Change My Life?");

        while(st.hasMoreTokens()) //hasMoreTokens 다음 문자열이 있는지 체크 함. true of false
            System.out.println(st.nextToken());//다음 토큰을 반환을 함

    }
}
```

9. 제네릭(generic), 컬렉션(collection)

1. 제네릭 프로그래밍(generic programming) *generic(포괄적인)

일반적인 코드를 작성하고 이 코드를 다양한 타입의 객체에 대하여 재사용하는 프로그래밍 기법

-> 데이터 형식에 의존하지 않고 하나의 값이 여러 다른 데이터 타입들을 가질 수 있는 방법

1) 제네릭클래스

1_하나의 코드로 여러가지 타입을 처리할 수 있다.

-> 어떤 타입이 들어오든 간에 그 타입으로 객체가 만들어진다.

2_ 제네릭 선언 방법 : class의 이름 옆에 "< >"를 붙이고 사이에 파라미터를 넣어준다.

사이에는 대문자 알파벳 한글자로 표시한다. ex) public class ClassName<T>{....}

2) 기존의 방법

일반적인 객체를 처리하려면 최상위 객체인 Object의 참조변수를 사용해야 한다.

-> Object로 값을 주고 받다보면 강제 형변환을 해주어야 하는 상황이 발생하고 성능이 좋지 않을 수 있다. 제네릭을 사용하면 기본틀을 가지고 새로만들어야하는 단점을 보완하고 다양한 타입의 객체에 대해서 재사용할 수 있도록 만들어준다.

2. 컬렉션(collection)

컬렉션은 자바에서 자료 구조를 구현한 클래스

*자료구조 : 일을 효율적으로 처리하게 만드는 알고리즘(리스트, 스택, 큐, 집합, 해쉬테이블 등)

1) 리스트(List) : 순서가 있는 데이터들의 집합

2) 스택(Stack) : 순서가 있는 데이터들의 집합을 LIFO(Last In First Out)의 방식으로 데이터를 관리하는 자료구조

3) 큐(Queue) : 순서가 있는 데이터들의 집합을 FIFO(First In First Out)의 방식으로 데이터를 관리하는 자료구조

4) 집합(Set) : 순서가 없는 데이터들의 집합(데이터들의 중복이 없음)

2-1. 컬렉션 인터페이스

모든 컬렉션의 상위 인터페이스로서 컬렉션이 가지고 있는 핵심 메서드 선언

인터페이스	설명
Collection	모든 자료 구조의 부모 인터페이스로서 객체의 모임을 나타낸다.
Set	집합(중복된 원소를 가지지 않는)을 나타내는 자료 구조
List	순서가 있는 자료 구조로 중복된 원소를 가질 수 있다.
Map	키와 값들이 연관되어 있는 사전과 같은 자료 구조
Queue	극장에서의 대기줄과 같이 들어온 순서대로 나가는 자료구조

9. 제네릭(generic), 컬렉션(collection)

분류	메소드	설명
기본 연산	int size()	원소의 개수 반환
	boolean isEmpty()	공백 상태이면 true 반환
	boolean contains(Object obj)	obj를 포함하고 있으면 true 반환
	boolean add(E element);	원소 추가
	boolean remove(Object obj)	원소 삭제
	Iterator<E> iterator();	원소 방문
벌크 연산	boolean addAll(Collection<? extends E> from)	c에 있는 모든 원소 추가
	boolean containsAll(Collection<?> c)	c에 있는 모든 원소가 포함되어 있으면 true
	boolean removeAll(Collection<?> c)	c에 있는 모든 원소 삭제
	void clear()	모든 원소 삭제
배열 연산	Object[] toArray()	컬렉션을 배열로 변환
	<T> T[] toArray(T[] a);	컬렉션을 배열로 변환

2) 인터페이스의 특징

1_List - 구현클래스 (Linked List, Stack, Vector, ArrayList) : 순서가 있는 데이터의 집합, 데이터의 중복을 허용한다.

2_Set - 구현클래스 (HashSet, TreeSet) : 순서를 유지하지 않는 데이터의 집합, 데이터의 중복을 허용하지 않는다.

3_Map - 구현클래스(HashMap, TreeMap, HashTable, Properties) : 키와 값의 쌍으로 이루어진 데이터의 집합. 순서 유지는 되지 않고, 키는 중복을 허용하지 않는다.

3) List 인터페이스

1_ ArrayList : 데이터가 삽입, 삭제 될 때마다 새로운 배열을 생성

2_ LinkedList : 데이터가 삽입, 삭제 될 때 해당 칸만 추가, 삭제하고 값의 위치가 변경

* 잦은 탐색 : ArrayList 유리, 잦은 수정: LinkedList유리

3. Set 인터페이스

집합(set)은 원소의 중복을 허용하지 않는다. 집합은 순서가 없고 중복을 허용하지 않는다.

1) HashSet

해쉬 테이블에 원소를 저장하기 때문에 성능면에서 가장 우수하다. 하지만 원소들의 순서가 일정하지 않은 단점이 있다. 해쉬는 중복이 없기때문에 주의해야한다.

4. Map 인터페이스

사전과 같은자료 구조, 키(Key)에 값(Value)이 매핑이된다.

-> 사전을 보면 알파벳 별로 홈이 있고 홈하나를 선택해서 열었을 경우에 그에 해당되는 단어들이 있다.

10. 예외처리(Exception)

1. 예외(Exception) : 런타입에러 (프로그램 실행 중에 어떤한 이유로 뻗는것)
2. 예외처리 : 프로그램이 뻗는 상황(런타입에러)이 왔을 때 어떻게 대응할지에 대한 명세를 작성하는것
- > 그러면 예외가 발생해도 프로그램이 뻗지않고 계속동작

3. 예외처리기의 개요(try -catch)

```
try{
  일단 실행할 문장
  }catch{
  예외가 발생하면 처리
}
```

-> try에서 오류가 발생하자마자 catch로 가서 예외처리를 함.

4. 예외의 종류

1) 에러 (Error): JVM을 돌리는 운영체제 자체에서 문제가 발생 // 심각한 에러
크게 나누자면 3가지 정도
1_OutOfMemoryError : 메모리가 부족할때 블루스크린 발생
2_AWTError : 표준입출력할때 모니터에서 발생
3_ThreadDeath

2) 예외 (Exception) 크게는 2가지 정도로 나뉨
① IOException : 밖으로 보내거나 안으로 들어올 때 발생하는 Exception
-> JVM이외의 메모리에 참고하고자 할때 발생을 하는 에러
-> 파일의 끝을 지나쳐서 읽으려고 하는 경우나 잘못된 URL을 사용하는경우
② RuntimeException
1_ClassCastException : 타입이 맞는 않을 경우
2_ArrayIndexOutofBoundsException : 배열의 범위가 벗어난 경우
3_NullPointerExceprion : 가르키고 있는 값이 비어있을 경우
4_ArithemeticException : 수학적인 오류
위와 같은 경우에는 개발자의 잘못이 대다수이며 선택적인 예외처리 가능(조건, 범위 등으로 처리 가능)

10. 예외처리(Exception)

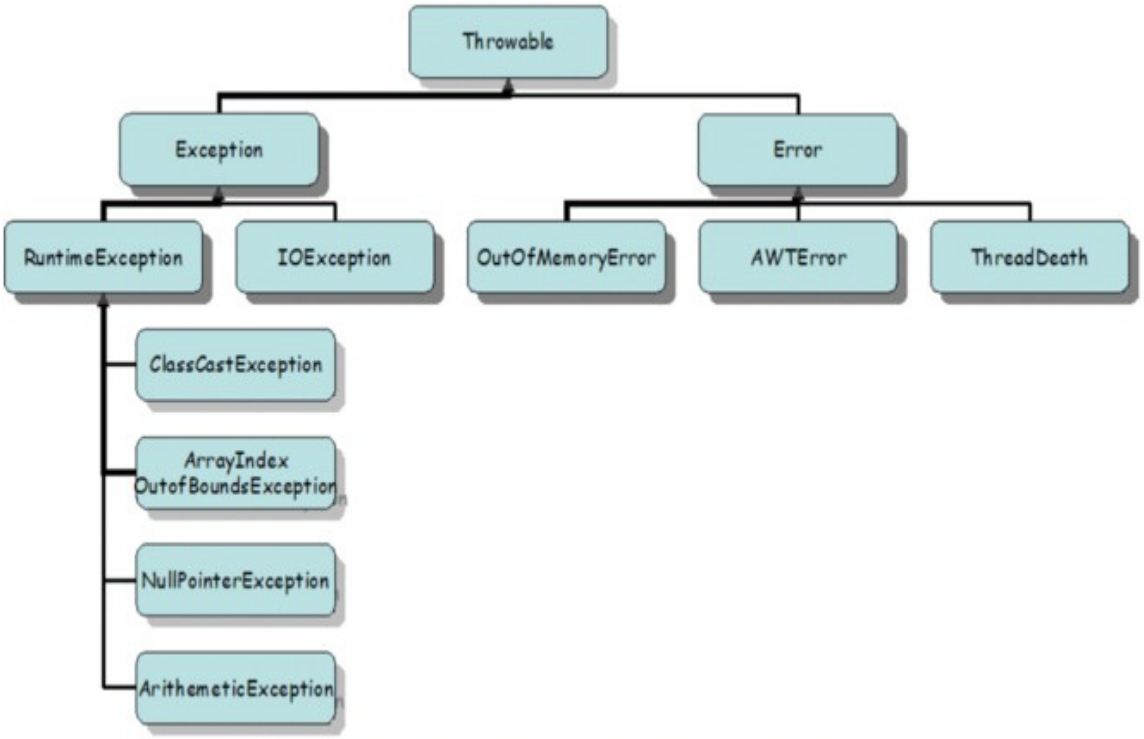


그림 15.4 일부 예외 관련 클래스의 계층도

5.체크예외와 비체크예외

자바 컴파일러는 RuntimeException은 확인하지않는다. 그 밖의 예외는 모두 처리를 확인한다.

1) 체크예외 : 컴파일러가 예외처리를 강제 -> IOException

① IOException이 발생했을 때 처리방법은 2가지이다.

1_ try-catch를 사용

-> try-catch 예외처리를 해당함수에서함(자동완성으로도 가능)

2_ 예외처리를 처리할 함수이름 옆에 throws IOException을 입력한다.

-> 해당함수의 몸통안에 IOException을 발생 시키는 코드를 예외처리 하지 않고 사용가능함.

대신 해당함수를 호출하는 녀석(사용하는 녀석)은 IOException에 대한 책임을 갖게됨

2) 비체크예외 : 컴파일러가 신경을 안쓰는 예외

6. 예외처리

1) 사용자 정의 예외생성하기

예외가 나오는 것처럼 사용자가 예외를 만들 수 있다.

Tip .. throws == 내가 책임을 갖고 있는게 아니라, 사용하는 놈이 책임

throw == 익셉션을 발생