객체지향 프로그래밍 - 기말

7주차(16.04.21)

❑ 오버로딩

* 한 클래스 안에서 같은 이름을 가진 함수가 여러개 있는 것
* 메소드의 이름이 동일해야함
* 인자의 개수가 서로 다르거나, 인자 타입이 서로 달라야함

```
// 메소드 오버로딩이 성공한 사례
class MethodOverloading {
   public int getSum(int i, int j) {
      return i + j;
   }
   public int getSum(int i, int j, int k) {
      return i + j + k;
   }
   public double getSum(double i, double j) {
      return i + j;
   }
}
```

```
// 메소드 오버로딩이 실패한 사례
class MethodOverloadingFail {
   public int getSum(int i, int j) {
      return i + j;
   }
   public double getSum(int i, int j) {
      return (double)(i + j);
   }
}
```

❑ this

- this 레퍼런스

* 현재 객체 자기 자신을 가리킴
  - 자기 자신에 대한 레퍼런스
  - 같은 클래스 내에서 클래스 멤버, 변수를 접근할 때 객체의 이름이 없으면 묵시적으로 this로 가정
  - 객체의 멤버 변수와 메소드 변수이 이름이 같은 경우
  - 객체 자신을 메소드에 전달 또는 반환할 때
* int id;

  void set(int x) {this.id = x;}

  void set(3); id 주소값이 3으로 바뀜
* this(), 같은 클래스의 다른 생성자 호출
* 생성자 내에서만 사용 가능 // 다른 메소드에서는 사용 불가

- this()
  - 같은 클래스의 다른 생성자 호출
  - 생성자 내에서만 사용 가능
    - 다른 메소드에서는 사용 불가
  - 반드시 생성자 코드의 제일 처음에 수행

```
public class Book {
   String title;
   String author;
   int ISBN;

   public Book(String title, String author, int ISBN) {
   this.title = title;
   this.author = author;
   this.ISBN = ISBN;
   }
   public Book(String title, int ISBN) {
      this(title, "Anonymous", ISBN);
   }
   public Book() {
      this(null, null, 0);
      System.out.println("생성자가 호출되었음");
   }
   public static void main(String [] args) {
      Book javaBook = new Book("Java JDK", "황기태", 3333);
      Book holyBible = new Book("Holy Bible", 1);
      Book emptyBook = new Book();
   }
}
```

title = "Holy Bible"
author = "Anonymous"
ISBN = 1

title = "Holy Bible"
ISBN = 1

❑ 객체의 치환

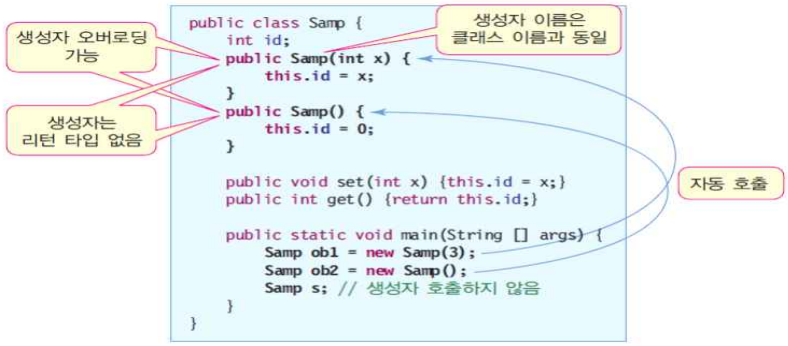
* 객체의 치환은 객체가 복사되는 것이 아니며 레퍼런스가 복사된다.

❑ 생성자

* 생성자의 특징
  - 생성자는 메소드 (new로 한 번만 부를 수 있다)
  - 생성자 이름은 클래스 이름과 동일
  - 생성자는 **new를 통해 객체를 생성할 때만 호출됨**
  - 생성자도 오버로딩 가능
  - 생성자는 리턴(return) 타입을 지정할 수 없다. **// 생성자의 주소만 리턴하기 때문.**
  - 생성자는 하나 이상 선언되어야 함

* 개발자가 생성자를 정의하지 않으면 자동으로 기본 생성자가 정의됨
  - 컴파일러에 의해 자동 생성 (아예 없을 때 생성)

생성자 정의와 생성자 호출



❑ 가비지 컬렉터

* 값을 바라보는 객체가 없으면 그 값은 가비지가 됨.

❑ 멤버 접근자

default (또는 package-private)	**•같은 패키지 내에서 접근 가능**
public	•패키지 내부, 외부 클래스에서 접근 가능
private	•정의된 클래스 내에서만 접근 가능 •상속 받은 하위 클래스에서도 접근 불가
protected	•같은 패키지 내에서 접근 가능 •다른 패키지에서 접근은 불가하나 상속을 받은 경우 하위 클래스에서는 접근 가능

멤버에 접근하는 클래스	멤버의 접근 지정자			
	default	private	protected	public
같은 패키지의 클래스	O	X	O	O
다른 패키지의 클래스	X	X	X	O

❑ static

* 종속되어 있는 클래스의 인스턴스들의 생성이나 활용을 도우는 목적으로 사용된다.

기말 시작

10주차(16.05.12)

* 인스턴스 : 객체의 구체적인 값
* 템플릿 : 양식, 서식, ex) 사람클래스(템플릿)을 가져다 씀
* 캡슐화 : 보여주는 필요한 데이터를 최소한으로 줄이고, 정보를 은닉 ex) TV
* static : 컴파일 시간에 만들어지는 것, 미리 메모리를 확보 되어 있는 것
* un-static : 실행 시간에 만들어 지는 것.

❑ super()

- 자식 클래스의 생성자 내에서는 부모클래스를 호출할 수 있음.
- 반드시 서브 클래스 생성자 코드의 제일 첫 라인에 와야함.

* 동적 바인딩 : 컴파일할 때 실행 됨, 실제 그 객체의 메소드를 쓰는 것,
  하위 클래스에 따라 결정 됨(그때그때 바인딩 됨)

❑ instanceof 연산자

- 객체가 뒤에 포함되는지 판별해주는 것. ex) "java" instanceof String

❑ abstract : 추상

* 추상 메소드 : 선언되어 있으나 구현되어 있지 않은 메소드
* 추상 클래스 : 추상 메소드를 하나라도 가진 클래스, 추상 메소드가 하나도 없지만 클래스 앞에 abstract로 선언한 경우
  - 객체를 생성할 수 없다. (객체가 될 수 없음)
  - 계층적 상속 관계를 갖는 클래스 구조를 만들 때
  - 설계와 구현을 분리 할 때 // ex) 한국인, 중국인, 미국인을 생성할 때 사람이지만 사람은 실제로 생성하지 않는다.

❑ interface : 다중 상속을 가능하게 함. // 면과 면사이 경계면

- 모든 메소드가 추상 메소드인 클래스
- 상수와 메소드만 갖는다.(필드는 없음)
- ex) public interface SerialDriver {...}
- 다중 상속이 안되는 이유가 어떤 것을 따라야하는지 몰라서 인데, interface는 아무것도 담고 있지 않기 때문에 상속가능
- 하위 클래스는 implements를 사용, implements를 사용할 때는 interface안의 메소드가 구현한 클래스에 있어야함.

```
//interface 예제, 플레이어를 쫓아다니는 몬스터//
import java.util.Scanner;

interface Character { //모든 캐릭터가 x, y가 무엇인지 알게 해줌
    int getX();
    int getY();
}

interface Controllable { //조작가능
    void move(int dir); // dir을 주면 따라 움직이는 것
}

interface Chaser { //따라다니게 만든 것
    void follow(int x, int y);
}

class player implements Character, Controllable {
    int x; int y;

    public int getX() { return x; } //// Character interface
    public int getY() { return y; } //// Character interface
    public void move(int i)      //// Controllable interface
    {
        switch(i) {
        case 1: x--; break;
        case 2: x++; break;
        case 3: y--; break;
        case 4: y++; break;
        }
    }

    public void setLocation(int x, int y) { //자기의 메소드임
        this.x = x; this.y = y;
    }
}
```

```
class monster implements Character, Chaser {
    int x; int y;

    public int getX() { return x; } //// Character interface
    public int getY() { return y; } //// Character interface
    public void follow(int x, int y)//// Chaser interface
    {
        int dx = x - this.x;
        int dy = y - this.y;
        this.x += (dx*2)*0.3;
        this.y += (dy*2)*0.3;
    }
    public void setLocation(int x, int y) {
        this.x = x; this.y = y;
    }
}

public class TestGame {

    public static void main(String[] arg) {
        Scanner scanner = new Scanner(System.in);
        player p = new player();
        monster m = new monster();
        p.setLocation(100, 100);
        m.setLocation(0, 50);
        while(true) {
            int px = p.getX();
            int py = p.getY();
            int mx = m.getX();
            int my = m.getY();
            System.out.println("player at (" + px + "," + py + ")");
            System.out.println("monster at (" + mx + "," + my + ")");

            System.out.print("player move:");
            int dir = scanner.nextInt();

            p.move(dir);
            m.follow(px, py);

System.out.println("----------------------------------");
        }
    }
}
```

❑ Packages

- 서로 관련된 클래스와 인터페이스의 컴파일 된 클래스(추상 클래스) 파일들을 하나의 디렉터리에 묶어 놓은 것
- 이름은 같지만 경로가 달라 서로 다른 파일로 취급 / ex) java scanner : import로 패키지 불러옴
- jar 파일로 압축할 수 있음

❑ API : 어플리케이션 프로그래밍 인터페이스

❑ OPEN GL API : 그래픽을 그려주는 API

❑ Object 클래스

- 모든 메소드가 추상 메소드인 클래스

메소드	설명
protected Object clone()	현 객체와 똑같은 객체를 만들어 반환. 오버라이딩 필요
boolean equals(Object obj)	obj가 가리키는 객체와 현재 객체가 비교하여 같으면 true 반환
Class getClass()	현 객체의 런타임 클래스를 반환
int hashCode()	현 객체에 대한 해쉬 코드 값 반환
String toString()	현 객체에 대한 스트링 표현을 반환
void notify()	현 객체에 대해 대기하고 있는 하나의 쓰레드를 깨운다.
void notifyAll()	현 객체에 대해 대기하고 있는 모든 쓰레드를 깨운다.
void wait()	현 객체의 다른 쓰레드가 notify() 또는 notifyAll() 메소드를 호출할 때까지 현 쓰레드를 대기하게 한다.

❑ Wrapper 클래스

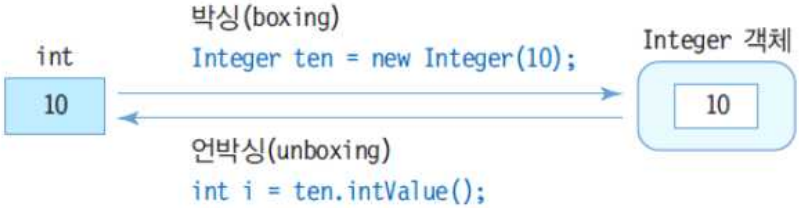
- 자바의 기본 데이터 타입을 클래스화한 8개의 클래스 // ex)Integer.parseInt
- 기본 데이터 타입을 사용할 수 없고 객체만 사용하는 컬렉션에 기본 데이터 타입을 Wrapper 클래스로 만들어 사용

기본 데이터 타입	byte	short	int	long	char	float	double	boolean
Wrapper 클래스	Byte	Short	Integer	Long	Character	Float	Double	Boolean

메소드	설명
static int bitCount(int i)	이진수 표현에서 1을 개수를 반환
float floatValue()	float 타입으로 변환된 값 반환
int intValue()	int 타입으로 변환된 값 반환
long longValue()	long 타입으로 변환된 값 반환
short shortValue()	short 타입으로 변환된 값 반환
static int parseInt(String s)	스트링을 10진 정수로 변환된 값 반환
static int parseInt(String s, int radix)	스트링을 지정된 진법의 정수로 변환된 값 반환
static Sting toBinaryString(int i)	이진수 표현으로 변환된 스트링 반환
static Sting toHexString(int i)	16진수 표현으로 변환된 스트링 반환
static Sting toOctalString(int i)	8진수 표현으로 변환된 스트링 반환
static Sting toString(int i)	정수를 스트링으로 변환하여 반환

ex) Integer I = new Integer(10); / int ii = I.intValue(); //ii=10 원래는 이렇게 사용해야함

* **박싱(boxing)** : 기본 데이터 타입을 Wrapper 클래스 객체로 변환
* **언박싱(unboxing)** : 기박싱의 반대 경우



❑ 스트링 객체

- 스트링 객체는 수정 불가능
- String s = new String("Hello"); / String t = s.concat("Java"); 일 때 Hello Java를 만드려면 String s는 변하지 않고 Hello Java를 String t에 넣어줌

○ ==과 equals()

* == : 변수를 비교
* equals() : 객체를 비교

○ int compareTo(String anotherString)

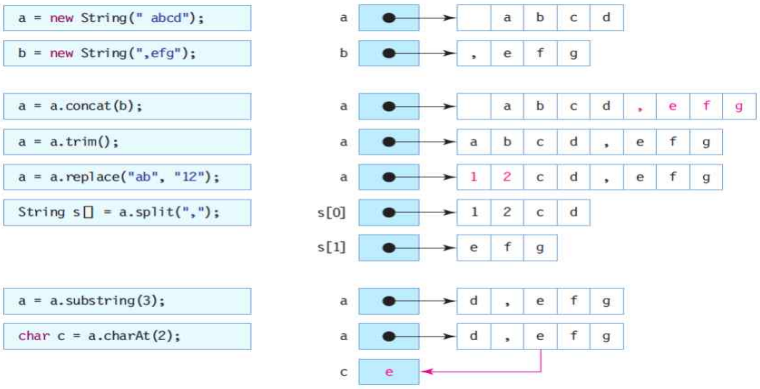
- 비교 연산자 ==는 레퍼런스를 비교하므로 문자열 비교에 사용 못함.
- 문자열이 같으면 0을 리턴

○ concat() : 새로운 문자열을 생성

- ex) String s1 = "abcd"; String s2 = "efgh"; -> s1 = s1.concat(s2); // s1에 abcdefgh가 저장 기존은 가비지

○ String.trim() : 공백제거

○ char charAt(int index) : 스트링에 포함된 문자 접근



○ StringBuffer : 임시로 저장해주는 것 (임시 저장소)

- append : 추가 / insert : 삽입

생성자	설명
StringBuffer()	문자를 포함하고 있지 않고 초기 크기가 16인 스트링 버퍼 생성
StringBuffer(charSequence seq)	seq가 지정하는 일련의 문자들을 포함하는 스트링 버퍼 생성
StringBuffer(int capacity)	문자를 포함하고 있지 않고 지정된 초기 크기를 갖는 스트링 버퍼 생성
StringBuffer(String str)	지정된 스트링으로 초기화된 스트링 버퍼 생성

메소드	설명
StringBuffer append(String str)	지정된 스트링을 스트링 버퍼에 덧붙인다.
StringBuffer append(StringBuffer sb)	지정된 스트링 버퍼를 스트링 버퍼에 덧붙인다.
int capacity()	현재 스트링 버퍼의 크기 반환
StringBuffer delete(int start, int end)	지정된 서브 스트링을 스트링 버퍼에서 제거
StringBuffer insert(int offset, String str)	지정된 스트링을 스트링 버퍼의 특정 위치에 삽입
StringBuffer replace(int start, int end, String str)	스트링 버퍼 내의 서브 스트링을 지정된 스트링으로 대치
StringBuffer reverse()	스트링 버퍼 내의 문자들을 반대 순서로 변경
void setLength()	스트링 버퍼 내 저장된 문자열 길이를 설정. 현재 길이보다 큰 경우 널 문자로 채우며 작은 경우는 문자열이 잘린다.

○ StringTokenizer : 하나의 스트링을 구분 문자로 분리하여 토근 형태로 파싱

- 끊을 때 한 덩어리를 토큰 (처리해야할 하나의 단위단위)
- ex) String query = "name=kitae&addr=seoul&age=21"; / StringTokenizer st = new StringTokenizer(quert, "&");

○ Math

- 기본적인 산술 연산을 수행하는 메소드 제공
- 모든 멤버 메소드는 static으로 정의됨 (객체를 만들어서 사용할 필요 없음)

메소드	설명
static double abs(double a)	절대값 반환
static double cos(double a)	cosine 값 반환
static double sin(double a)	sine 값 반환
static double tan(double a)	tangent 값 반환
static double exp(double a)	e^a 값 반환
static double ceil(double a)	지정된 실수보다 크거나 같은 수 중에서 가장 작은 정수를 실수 타입으로 반환
static double floor(double a)	지정된 실수보다 작거나 같은 수 중에서 가장 큰 정수를 실수 타입으로 반환
static double max(double a, double b)	두 수 중에서 큰 수 반환
static double min(double a, double b)	두 수 중에서 작은 수 반환
static double random()	0.0보다 크거나 같고 1.0보다 작은 임의의 수 반환
static double rint(double a)	지정된 실수와 가장 근접한 정수를 실수 타입으로 반환
static double round(double a)	지정된 실수를 소수 첫째 자리에서 반올림한 정수를 실수 타입으로 반환
static double sqrt(double a)	제곱근을 반환

○ Calendar : 객체 생성 및 날짜와 시간

- Calendar는 추상 클래스이므로 new Calendar() 하지 않음 // Calendar now = Calendar.getInstance(); 이용

필드	의미	필드	의미
YEAR	년도	DAY	한 달의 날짜
MONTH	달	DAY_OF_WEEK	한 주의 요일
HOUR	0 ~ 11시로 표현한 시간	AM_PM	오전인지 오후인지 구분
HOUR_OF_DAY	24시간을 기준으로 한 시간	MINUTE	분
SECOND	초	MILLISECOND	밀리초

시콘드 리스트

- 랜덤으로 찾아 주는 것

셋 : 집합의 순서가 없는 것 // {1, 2, 3}이나 {2, 3, 1}이나 같다.
스택 : 나중에 들어온 순서대로 데이터를 빼는 것
큐 : 먼저 들어온 순서대로 데이터를 빼는 것

```java
//달의 현재 상태 구하기// 입력받은 날의 달 상태 구하기 //
package test01;

import java.util.*;

public class test01 {
	public static void main(String[] arg){
		Scanner scan = new Scanner(System.in);
		Calendar curTime = Calendar.getInstance();
		Calendar firstDate = Calendar.getInstance();

		firstDate.set(1900,0,31); // 달은 0부터 시작해서 12월이지만 11입력

		int year, month, day;
	year = curTime.get(Calendar.YEAR);
	month = curTime.get(Calendar.MONTH);
	day = curTime.get(Calendar.DAY_OF_MONTH);
	System.out.println("오늘은 " + year + "년 " + (month+1) + "월 " + day + "일");


		long daysPassed = daysBetween(firstDate, curTime);
		double moon = moonPhase(daysPassed);

		System.out.println("달의 현재 상태 = " + moon);
		////////////////////////////////////////////////////////////////

		System.out.println("달의 상태를 알고 싶은 날짜를 입력 : ");
		String Days = scan.nextLine();
		String data[] = Days.split(". ");

		int[] arr = new int[data.length];

		for(int i=0; i<data.length; i++)
		{
			arr[i] = Integer.parseInt(data[i]);
		}

		curTime.set(arr[0], arr[1]-1, arr[2]);

		daysPassed = daysBetween(firstDate, curTime);
		moon = moonPhase(daysPassed);
		System.out.println("달의 현재 상태 = " + moon);
	}
	//달의 현재 상태 구하기
	static double moonPhase(long daysPassed) {
		double cycle = 29.530508853;
		double phase = daysPassed/cycle;
		phase -= Math.floor(phase); // 현재 상태
		return phase;
	}
	//보름달에서 부터의 날짜 구하기
	public static long daysBetween(Calendar s, Calendar e) {
		Calendar date = (Calendar) s.clone(); // 생일을 복제해서 씀, 오브젝트를 리턴
		int days = 0;
		while(date.before(e)) {
			date.add(Calendar.DAY_OF_MONTH, 1);
			days ++;
		}
		return days;
	} // s는 태어난 날짜, e는 현재 날짜
}
```

Calendar date = (Calendar) s.clone(); 설명

사람클래스(병규)와 동물 클래스(배성)가 있는데

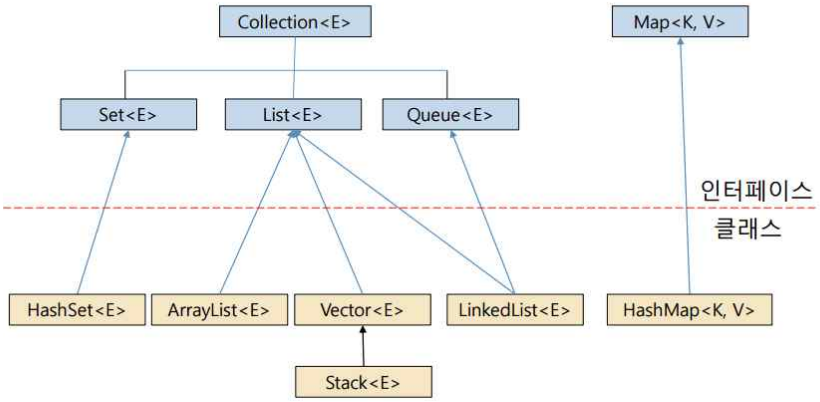
배성이의 데이터를 병규한테 넣으려면, 배성이를 복제해서 다운캐스팅을 하면 되는데,

타입자체가 오브젝트를 리턴해야하는데 자동으로 다운캐스팅할 수 없어서 (Calendar)로 표시를 해줌

❑ 컬렉션

○ 컬렉션(collection) // C++의 템플릿(Templeate)과 동일(모든 종류의 데이터 타입을 다룰 수 있도록 클래스나 메소드를 작성하는 기법)

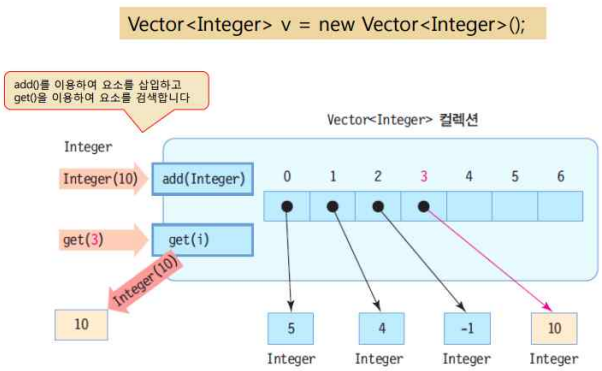
- 객체의 개수를 염려할 필요가 없다
- 링크드 리스트, 벡터 등이 이미 구현되어 있다.
- 요소(element)라고 불리는 가변 개수의 객체들의 모음
- 다양한 객체들의 삽입, 삭제, 검색 등을 관리하기 용이
- 객체만 사용가능 / 변수 사용x / Int, Double 등으로 받을 수 있음



<E> : 엘리멘탈 , <K> : 키 값 , <V> 밸류 값

○ Vector<E> : <E>에 타입 설정 // add(E e)를 하면 모든 타입이 들어올 수 있음

- 배열의 길이가 자동으로 설정



메소드	설명
boolean add(E e)	벡터의 맨 뒤에 요소 추가
void add(int index, E element)	지정된 인덱스에 지정된 객체를 삽입
int capacity()	벡터의 현재 용량 반환
boolean addAll(Collection<? extends E> c)	c가 지정하는 컬렉션의 모든 요소를 벡터의 맨 뒤에 추가
void clear()	벡터의 모든 요소 삭제
boolean contains(Object o)	벡터가 지정된 객체를 포함하고 있으면 true 반환
E elementAt(int index)	지정된 인덱스의 요소 반환
E get(int index)	지정된 인덱스의 요소 반환
int indexOf(Object o)	지정된 객체와 같은 첫 번째 요소의 인덱스 반환. 없으면 -1 반환
boolean isEmpty()	벡터가 비어있으면 true 반환
E remove(int index)	지정된 인덱스의 요소 삭제
boolean remove(Object o)	지정된 객체와 같은 첫 번째 요소를 벡터에서 삭제
void removeAllElements()	벡터의 모든 요소를 삭제하고 크기를 0으로 만듬
int size()	벡터가 포함하는 요소의 개수 반환
Object[] toArray()	벡터의 모든 요소를 포함하는 배열을 반환

// 자바는 박싱/언박싱 기능 자동 지원(Wrapper 클래스)()

```
import java.util.Vector;

public class aaaa {

	public static void main(String[] s) {
		// 벡터 생성
		Vector<Integer> v = new Vector<Integer>();
		// <Integer>을 쓰지 않으면 오브젝트형으로 관리됨

		v.add(5);
		v.add(4);
		v.add(01);

		// 벡터 중간에 삽입하기
		v.add(2, 100); // 4와 -1 사이에 정수 100 삽입

		System.out.println("벡터 내의 요소 객체 수 :" + v.size());
		System.out.println("벡터의 현재 용량 :" + v.capacity());

		// 모든 요소 정수 출력하기
		for (int i = 0; i < v.size(); i++) {
			int n = v.get(i);
			System.out.println(n);
		}
	}
}
```

```
<terminated> aaaa [Java App
벡터 내의 요소 객체 수 :4
벡터의 현재 용량 :10
5
4
100
1
```

○ ArrayList<E> : 스레드 자동 동기화 지원이 없음

* 스레드 : 프로세스 안에 각각의 일을 하는 것(), 병렬처리를 하기 위한 것
* 동기화 : 스레드의 시간을 맞추는 것
* 락 : 스레드 들이 움직일 때 하나의 스레드가 쓰고 있을 때 다른 스레드가 못쓰게 하는 것

○ Iterator<E> :반복자

- Vector<E>, ArrayList<E>, LinkedList<E>가 상속받는 인터페이스
  - 리스트 구조의 컬렉션에서 요소의 순차 검색을 위한 메소드 포함
- Iterator<E> 인터페이스 메소드

메소드	설명
boolean hasNext()	다음 반복에서 사용될 요소가 있으면 true 반환
E next()	다음 요소 반환
void remove()	마지막으로 반환된 요소 제거

```
//Point 클래스의 객체들만 저장하는 벡터 만들기// Vector, Iterator 사용
import java.util.Iterator;
import java.util.Vector;

class Point {
	private int x;
	private int y;

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

	public String toString() {
		return "("+x+","+y+")";
	}
}

public class PointVector {

	public static void main(String[] arg) {

		Vector<Point> v = new Vector<Point>();

		v.add(new Point(1,2));
		v.add(new Point(2,2));
		v.add(new Point(3,2));
		v.add(new Point(4,2));

		v.insertElementAt(new Point(0,0), 2);

		Iterator<Point> it = v.iterator();
		while(it.hasNext()){
			Point n = it.next();
			System.out.println(n);
		}
	}
}
```

```
Console  Tasks
<terminated> aaaa [Java Ap
(1,2)
(2,2)
(0,0)
(3,2)
(4,2)
```

```
//Point 클래스의 객체들만 저장하는 벡터 만들기// Vector, Iterator 사용
import java.util.*;

class Point {
	private int x;
	private int y;

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

	public String toString() {
		return "("+x+","+y+")";
	}

	public int getX() { return x; }
	public int getY() { return y; }
}

public class PointVector2 {

	public static void main(String[] arg) {

		Vector<Point> v = new Vector<Point>();

		v.add(new Point(1,2));
		v.add(new Point(2,2));
		v.add(new Point(3,2));
		v.add(new Point(4,2));

		v.insertElementAt(new Point(0,0), 2);

		Iterator<Point> it = v.iterator();
		double X = 0.0; double Y = 0.0;
		int n=0;
		while(it.hasNext()) {
			Point p = it.next();
			X += p.getX();
			Y += p.getY();
			n++;
		}
		X /= n; Y /= n;
		System.out.println("Center of Mass = (" + X + "," + Y + ")");

	}
}
```

```
Problems  Javadoc  Decla
<terminated> PointVector2 [Java App
Center of Mass = (2.0,1.6)
```

○ HashMap<K,V> : 원래 형태를 알 수 없게 만든 것(아이디, 패스워드에 많이 쓰임)

- hash함수 : 원래의 데이터를 알 수 없는 수로 으깨어 놓는 것 (암호에 많이 쓰임) // 역함수
- 키와 값이 한 쌍으로 삽입됨
- 요소 검색 : get() 메소드 / 요소 삽입 : put() 메소드
- Set<String> keys = javaScore.keyset(); 으로 Iterator할 수 있음

- java.util.HashMap
  - K는 키로 사용할 요소의 타입을, V는 값을 사용할 요소의 타입 지정
- 키(key)와 값(value)의 쌍으로 구성되는 요소를 다루는 컬렉션
  - 키와 값이 한 쌍으로 삽입됨
  - 키는 내부적으로 해시맵에 삽입되는 위치 결정에 사용
  - 값을 검색하기 위해서는 반드시 키 이용
- 삽입 및 검색이 빠른 특징
  - 요소 검색 : get() 메소드
  - 요소 삽입 : put() 메소드
- HashMap<String, String> 생성, 요소 삽입, 요소 검색

```
HashMap<String, String> h = new HashMap<String, String>();
h.put("apple", "사과"); // "apple" 키와 "사과" 값의 쌍을 해시맵에 삽입
String kor = h.get("apple"); // "apple" 키로 값 검색. kor는 "사과"
```

메소드	설명
void clear()	HashMap의 모든 키 삭제
boolean containsKey(Object key)	키를 포함하고 있으면 true 리턴
boolean containsValue(Object value)	하나 이상의 키를 지정된 값에 매핑시킬 수 있으면 true 리턴
V get(Object key)	지정된 키에 맵핑되는 값을 리턴하거나 맵핑되는 값이 없으면 null 리턴
boolean isEmpty()	HashMap이 비어 있으면 true 리턴
Set<K> ketSet()	HashMap에 있는 모든 키를 담은 Set<k> 컬렉션 리턴
V put(K key, V value)	key와 value를 매핑하여 HashMap에 저장
V remove(Object key)	지정된 키와 이에 매핑된 모든 값들을 HashMap에서 삭제
int size()	HashMap에 포함된 요소의 개수 리턴

```
// HashMap<K,V>로 사전 만들기 //
import java.util.*;

public class hashmap1 {

    public static void main(String[] s) {
        Scanner myScanner = new Scanner(System.in);

        HashMap<String, String> dictionary = new HashMap<String,String>();

        while(true) {
            System.out.println("input key:");
            String key = myScanner.next();
            if(key.equals("end")) break;

            System.out.println("meaning: ");
            String value = myScanner.next();

            dictionary.put(key, value);
        }

        // 모든 사전 데이터 출력

     //   1. set<String> keySet <- dictionary
        Set<String> keySet = dictionary.keySet();

     //   2. Iterator <- keySet
        Iterator<String> keyIt = keySet.iterator();

     //   3. Iterator:next으로 get()
        while(keyIt.hasNext()){
                String curKey = keyIt.next();
                System.out.println(dictionary.get(curKey));
        }

        ////////////////////////////////////////
        while(true) {
                System.out.println("search key:");
                String key = myScanner.next();
                if(key.equals("end")) break;
                System.out.println("meaning : " + dictionary.get(key));
        }
    }
}
```

○ Collections // 컬렉션에 대해 연산을 수행하고 결과로 컬렉션 리턴

- ex) Collections.sort(myList); / Collections.reverse(myList);
- 컬렉션에 포함된 요소들을 소팅(정렬)하는 sort() 메소드
- 요소의 순서를 반대로 하는 reverse() 메소드
- 요소들의 최대, 최솟값을 찾아내는 max(), min() 메소드
- 특정 값을 검색하는 birnarySearch() 메소드

* 이진탐색 : 매우 강력한 탐색 기법 // 찾고자 하는 것을 반으로 나눠서 탐색

○ Stack

* 과제 : 스택 만들기!!!! / 자신만의 클래스를 집어 넣어 보기

예제 7-9 : 제네릭 스택 만들기

스택 자료 구조를 제네릭 클래스로 선언하고, String과 Integer형 스택을 사용하는 예를 보여라.

```
class GStack<T> {
   int tos;
   Object [] stck;
   public GStack() {
      tos = 0;
      stck = new Object [10];
   }
   public void push(T item) {
      if(tos == 10)
         return;
      stck[tos] = item;
      tos++;
   }
   public T pop() {
      if(tos == 0)
         return null;
      tos--;
      return (T)stck[tos];
   }
}
```

```
public class MyStack {
   public static void main(String[] args) {
      GStack<String> stringStack = new GStack<String>();
      stringStack.push("seoul");
      stringStack.push("busan");
      stringStack.push("LA");

      for(int n=0; n<3; n++)
         System.out.println(stringStack.pop());

      GStack<Integer> intStack = new GStack<Integer>();
      intStack.push(1);
      intStack.push(3);
      intStack.push(5);

      for(int n=0; n<3; n++)
         System.out.println(intStack.pop());
   }
}
```

```
LA
busan
seoul
5
3
1
```

제네릭에서 배열의 제한

- 제네릭 클래스 또는 인터페이스의 배열을 허용하지 않음

```
GStack<Integer>[] gs = new GStack<Integer>[10];
```

- 제네릭 타입의 배열도 허용되지 않음

```
T[] a = new T[10];
```

  - 앞 예제에서는 Object 타입으로 배열 생성 후 실제 사용할 때 타입 캐스팅

```
return (T)stck[tos]; // 타입 매개 변수 T타입으로 캐스팅
```

- 타입 매개변수의 배열에 레퍼런스는 허용

```
public void myArray(T[] a) {....}
```

예제 7-10 : 스택의 내용을 반대로 만드는 제네릭 메소드 만들기

42

예제 7-9의 GStack을 이용하여 주어진 스택의 내용을 반대로 만드는 제네릭 메소드 reverse()를 작성하라.

```
public class GenericMethodExample {
   // T가 타입 매개 변수인 제네릭 메소드
   public static <T> GStack<T> reverse(GStack<T> a) {
      GStack<T> s = new GStack<T>();
      while (true) {
         T tmp;
         tmp = a.pop(); // 원래 스택에서 요소 하나를 꺼냄
         if (tmp==null) // 스택이 비었음
            break;
         else
            s.push(tmp); // 새 스택에 요소를 삽입
      }
      return s; // 새 스택을 반환
   }
```

```
   public static void main(String[] args) {
      // Double 타입의 GStack 생성
      GStack<Double> gs =
            new GStack<Double>();

      // 5개의 요소를 스택에 push
      for (int i=0; i<5; i++) {
         gs.push(new Double(i));
      }
      gs = reverse(gs);
      for (int i=0; i<5; i++) {
         System.out.println(gs.pop());
      }
   }
}
```

```
0.0
1.0
2.0
3.0
4.0
```

❑ 스트림 : 강이라는 뜻. 1자로 이어져 있음

* 바이트 스트림 : 문자 이외에도 읽어 들임 (바이너리 파일 : 문자열 포함 모든 것) // ex) InputStream
* 문자 스트림 : 문자열만 읽어 들임 //ex) Scanner

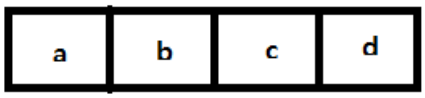
○ FileInputStream : 파일을 읽어 들임

○ FileOutputStream : 파일을 사용

- FileOutputStream four = new FileOutputStream("c:\\test.out")

```
import java.io.*;

public class FileStream {
	public static void main(String[] arg){

		try{
			FileOutputStream fout = new FileOutputStream("./test.out");

			for(int i=0; i<=1024; i++){
				int num=i;
				fout.write(num);
				fout.write(num>>8);
				fout.write(num>>16);
				fout.write(num>>24);
			}
			fout.close();

			FileInputStream fin = new FileInputStream("./test.out");
			int input;
			while((input = fin.read())!=-1){
				int value =0;
				value +=input;
				value += fin.read()*256;
				value += fin.read()*256*256;
				value += fin.read()*256*256*256;
				System.out.println(value);
			}
			fin.close();
		}

		catch(IOException e){
			System.out.println("IO Error");
		}
	}
}
```

정수는 4바이트/ 1바이트 : 0~255

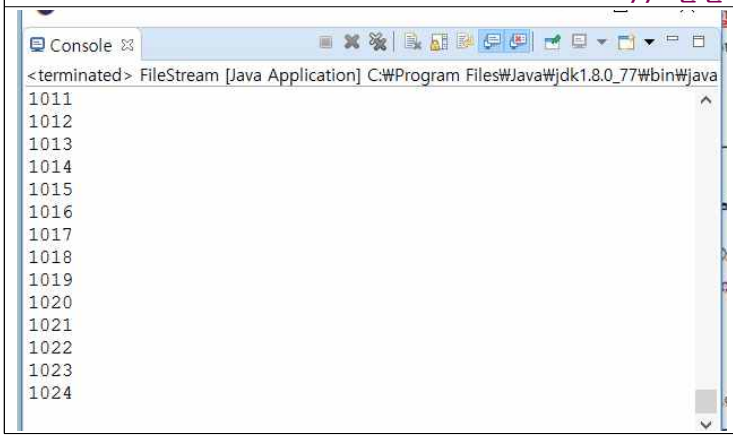


d + c*256 + b*256*256 + a*256*256*256

d만 출력하기 때문에 fout.write(num); 밑에 num/=256; 와

value += fin.read()*256;를 해줌

// 간단하게 >> 쉬프트연산으로 바꿀 수 있다.



* Encoding : 암호화 하다.

|	둘중 하나가 참	1 | 0 , 1 | 1...	
&	둘 다 참	1 & 1	
^	하나만 참이어야함	1 ^ 0 , 0 ^ 1	^를 한번 더 하면 원래대로 돌아옴

○ argv : 3개, 0번 읽기, 1번 복사, 2번

```
import java.io.*;

public class MyDiary {
    public static void main(String[] argv) {
        try {

            if(argv.length<3) {
                System.out.println("arguments missing");
                return;
            }
            FileInputStream fin = new FileInputStream(argv[0]); // source file //[0]은파일명
            FileOutputStream fout = new FileOutputStream(argv[1]); // destination file [1]은 생성할 파일 명
            int pass = Integer.parseInt(argv[2]); //[2]는 암호
            int c;
            while ( (c=fin.read()) != -1 ) {
                fout.write(c^pass);
            }
            fin.close();
            fout.close();
        }
        catch (IOException e) {

        }
    }
}
```

○ File클래스 : 파일 이동, 삭제 등을 하는 클래스 // import java.io.File

암호를

type myDiary.txt : 파일 o

Java FileEncoding ./myDiary.txt

```
import java.io.*;

public class MyDiary {
    public static void main(String[] argv) {
        try {
            if(argv.length<2) {
                System.out.println("arguments missing");
                return;
            }
            String tempFile = "./tempXYZEncoded_YYYY.out";
            FileInputStream fin = new FileInputStream(argv[0]); // source file
            FileOutputStream fout = new FileOutputStream(tempFile); // destination file
            int pass = Integer.parseInt(argv[1]);
            int c;
            while ( (c=fin.read()) != -1 ) {
                fout.write(c^pass);
            }
            fin.close();
            fout.close();

            // remove original and rename temp file
            File original = new File(argv[0]);
            File newFile = new File(tempFile);

            original.delete();
            newFile.renameTo(original);
        }
        catch (IOException e) {
        }
    }
}
```

중간고사 예제

1) 객체지향 프로그래밍이란 무엇인가?

- 객체 지향 프로그래밍이란, 해결할 과제를 실제 세상의 객체와
  객체 간의 상호 관계로 모델링하여 인간의 사고에 가깝게 표현한 프로그래밍이다.

2) 자바가상기계(JVM, Java Virtual Machine)란 무엇이며, 이것 때문에 자바(Java)가 다른 언어와 어떤 차이점을 갖게 되는가?

-자바가상기계란, 자바로 작성된 응용프로그램을 윈도우나 유닉스와 같은 컴퓨터 운영체계에서 원활히 운용될 수 있도록 하는 소프트웨어이다.
/ C/C++같은 기존 언어는 플랫폼 종속적이지만 자바는 WORA로 어느 플랫폼에서나 실행가능하다.

3) JDK와 JRE의 차이는 무엇인가?

- JDK는 자바 프로그램 개발을 위해 설치하는 환경이고,
  JRE는 일반 사용자들이 자바 실행을 위해 사용하는 환경이다.

4) 두 개의 정수를 입력 받아 큰 수를 출력하는 프로그램을 자바(Java)로 작성하라.

- 아래에 작성

5) 값에 의한 호출(call by value)와 참조에 의한 호출(call by reference)의 차이를 설명하고, 자바(Java)에서 호출은 어떻게 이루어지는지 설명하라.

- call by value는 함수를 호출할 때 단순히 값을 전달하는 함수 호출이고,
  call by reference는 메모리의 접근에 사용되는 주소 값을 전달하는 형태의 함수 호출을 말한다.
  call by reference(주소 값)을 사용하는 이유는 메모리 사용을 줄이기 위해서이다.

[int형 배열 10만개를 저장하면 메모리가 40만 바이트가 필요한데, 주소 값을 사용하면 하나만 가지면 된다.]

자바에서는 call by reference(주소값)는 구현되지 않는다. 따라서 call by value 호출을 사용하며, 기본형타입과 참조형타입은 인수를 넘길 때 값 복사가 일어난다. 즉, 참조 값끼리 아무리 복사를 하더라도 객체 내부의 메모리끼리의 복사는 이루어지지 않기 때문에, 단지 참조 값만이 복사되어 호출된다.

// 자바에서는 call by value만이 존재한다. 기본 타입은 말 할 것도 없고, 참조타입은 call by reference와 비슷하지한 효과가 구현되지만, 엄밀히 말하면 call by value이다.

6) 클래스(Class)와 객체(instance), 그리고 메소드(method)에 대해 설명하라.

- 클래스 : 사물의 특성을 소프트웨어적으로 추상화하는 설계도.(객체에 대한 정의)
  객체 : 사물의 특성을 소프트웨어적으로 추상화한 것(사물 또는 개념)(클래스의 인스턴스(실체))
  메소드 : 클래스를 수행하는 기능

7) 오버라이딩(overriding)은 무엇이며, 오버로딩(overloading)과 어떻게 다른지 설명하라. 간단한 두 개의 클래스 class A와 class B를 실제로 만들고, 이 두 클래스에 메소드를 자바 코드로 작성하여 오버로딩(overloading)과 오버라이딩(overriding)을 표현해 보라.

- 오버라이딩 : 기존에 있던 것을 덮어 쓴 것.(재정의하는 것)
  오버로딩은 : 여러개의 동작들을 만든 것

8) 정적(static) 메소드는 어떤 특성을 갖는가?

- static 멤버는 non-static멤버와 달리 이미 외부에 별도로 존재하며 이를 공유한다.
  (new를 써서 객체가 만들어지기 전에 존재하여야 한다)
- static 메소드는 오직 static멤버만 접근이 가능하다.

- 종속되어 있는 클래스의 인스턴스들의 생성이나 활용을 도우는 목적으로 사용된다.

- 클래스 내에서 인스턴스끼리 공유하는 변수이다.(공유 변수)

- 메서드에 static을 붙여 사용하는 경우 메서드의 기능이 인스턴스의 상태와 상관없이 수행 할 수 있는 경우이다.

- static 클래스가 로딩 된 시점에서 메모리 공간을 가지고 있기 때문에
  인스턴스 생성 없이 변수나 상수나 메소드를 클래스 이름 뒤에 참조연산자 .(마침표)를 이용해서 바로 접근이 가능하게 된다. ex) StaticInner.Inner = new StaticInner.Inner();

```
>> 다중 상속을 써야할 경우 interface(implements)를 쓴다.
// inter 파일 //
/*iSender 인터페이스를 사용할 경우 iSender에 있는 모든 메소드를 구현하지 않으면
 추상 클래스가 되어 class에 abstract를 붙여줘야 한다. 아래 클래스inter 에서 다른 인터페이스
 void Send(String strReceiver)와 void Receive(String strSender)가 구현되어 있지
 않으므로 abstract를 붙여 추상클래스로 선언했다. */
public abstract class inter extends CChildclass implements iSender, iReceiver
{
	String strSender;
	String strReceiver;

	public inter()
	{
	}

	public void SetSender(String strSender)
	{
		this.strSender = strSender; //this는 현재 클래스의 멤버 변수 strSender이며,
		 //값으로 쓰인 strSender는 SetSender함수의 Argument인자인 strSender이다.
	}

	public void SetReceiver(String strAreceiver)
	{
		this.strReceiver = strAreceiver;
	}
}
```

```
// iSender 파일 //
public interface iSender {
	void SetSender(String strSender);
	void Snd(String strReceiver);
}
```

```
// iReceiver 파일 //
public interface iReceiver{
	void SetReceiver(String strReceiver);
	void Receive(String strSender);
}
```

```
// CChildclass파일 //
public class CChildclass {
}
```

다른 사람이 만든 코드를 받아올 때 맞춰서 해주는 것

자바 : 제너릭

C++ : 템플릿

// Action 이벤트를 이용한 콤보 박스 활용 예 / 콤보박스를 만들고 콤보박스를 선택한 애를 이미지 출력

코드밑에 약간의 문답형 문제 / 많으면 4문제 / 오픈북 다음주 10시

1공 309호

과제 : 8-2의 달 위상 프로그램을 개선한다. 현재 입력을 연도, 달, 일자로 받는데(토큰),
이를 2011.4.23. 과 같이 일괄적으로 받아들여 StringToken를 활용하여 입력을 처리할 수 있도록 고쳐라.
가산점 : 2011.4.23. 뿐만 아니라 2011. 4. 23처럼 띄워쓰거나 20110423 같이 다른 형식도 써보면 가산점.
2011. 04. 23같이 표현도 처리할 수 있도록

```
// 과제 : 내가 짤 수 있는 가장 훌륭한 자바 프로그램!!! 내면 무조건 A. //
// Java 과제3 파일 //
턴제카드 게임 과제
```

```
//Calendar을 이용해 날짜 출력해보기(태어난 날부터 현재까지 날짜 계산)// 무식한 방법.
import java.util.Calendar;

public class Calendar {
	public static void main(String[] arg){
		Calendar curTime = Calendar.getInstance();
		Calendar birthDay = Calendar.getInstance();

		birthDay.set(1993,11,06); // 달은 0부터 시작해서 12월이지만 11입력

		long daysPassed = daysBetween(birthDay, curTime);

		System.out.println(daysPassed);
	}


	public static long daysBetween(Calendar s, Calendar e) {
		Calendar date = (Calendar) s.clone(); // 생일을 복제해서 씀, 오브젝트를 리턴
		int days = 0;
		while(date.before(e)) {
			date.add(Calendar.DAY_OF_MONTH, 1);
			days ++;
		}
		return daysBetween;
	} // s는 태어난 날짜, e는 현재 날짜
}
```