2주차(3)

추상 메서드와 추상 클래스

work / 0415 / src / ex5

추상메서드의 구성 : [접근제한] abstract [반환형] [메서드명](); //{ }없이 ;으로 마무리됨
abstract [접근제한] [반환형] [메서드명](); //{ }없이 ;으로 마무리됨
완성된 단계가 아닌 미완성적 개념으로 두고, 나중에 타 클래스 내에서 재정의하여
사용할 수 있도록 만드는 것.

추상메서드가 한 개 이상 정의되어 있는 클래스를 추상 클래스라고 하는데,
추상 클래스 또한 abstract를 통해 자신이 추상클래스임을 명시해줘야 한다.
추상클래스의 구성 : [접근제한] abstract class [클래스명]{ }

예제로 한번 살펴봐요

AbsEx정의(abstract)

```
abstract public class AbsEx {
	// 추상 클래스는 일반 클래스와 같다.
	// 즉, 변수,상수, 일반메서드 모두 가질 수 있다는 뜻이다.
	// 추상 클래스라는 것은
	// 추상메서드를 하나라도 가진 클래스를 의미한다.

	int value = 100;
	final int VALUE2 = 1000; // 상수

	public int getValue(){ // 일반적인 메서드
		return value;
	}

	//추상 메서드는 body가 없다. - abstract로 시작한다.
	abstract public void setValue(int n);

	// 추상 메서드는 body가 없기때문에 이를 "미완성적 개념"이라 한다.
	// 그러므로 이 미완성적 개념을 자식이
	// 받아서 완성 시켜야하는 것이 하나의 조건인 것이다.
}
```

AbsChild클래스 정의

```
public class AbsChild extends AbsEx{
```

```
	int age = 10;

	//추상 클래스를 상속받은 자식 클래스는
	//부모인 추상클래스가 가지고 있는 추상 메서드를 무조건 받아두어야 한다.
	//재정의 할 필요는 없지만 오버라이딩 해서 가지고는 있어야 한다는 의미.
	@Override
	public void setValue(int n) {
		age += n;
	}

	@Override
	public int getValue() {
		return age;
	}
}
```

AbsMain정의

```
public class AbsMain {
	public static void main(String[] args) {
		//추상클래스는 인스턴스를 직접 가질 수 없다.
		//즉, 객체화 시킬 수 없다는 것.
		AbsEx a1 = new AbsEx(); //오류확인 후 주석

		// 그러므로 추상클래스는 자신의 기능을 자식이 완성 시키도록 조건부
		// 상속하여 자식클래스가 생성될 때 객체화된다.
		AbsChild a1 = new AbsChild();
		a1.setValue(20);
		System.out.println(a1.getValue());
	}
}
```

----------예제 1

AbsClass정의

```
public abstract class AbsClass {
	int value = 100;

	public int getValue(){
		return value;
	}

	public abstract int changeValue(); // 추상메서드
```

```
}
```

AbsChild1정의

```
public class AbsChild1 extends AbsClass{

	@Override
	public int changeValue() {
		return value += 10;
		//value는 부모의 멤버. 상속 받았기 때문에 사용 가능
	}
}
```

AbsChild2정의

```
public class AbsChild2 extends AbsClass{

	@Override
	public int changeValue() {
		return value -= 3;
	}
}
```

AbsMain정의

```
public class AbsMain {

	public static void main(String[] args) {
		// 추상클래스는 객체화 할 수 없다.
		AbsClass a1 = new AbsClass(); //확인 후 주석

		//추상 클래스는 자식클래스에 의해 객체화 된다.
		AbsChild1 ac1 = new AbsChild1();
		System.out.println(ac1.changeValue());
		↑여기까지 일단 결과로 110이 나오는 이유먼저 설명

		// 그럼 test라는 메서드를 한 개 만들어 볼게요
		// 이때 아래에 있는 test(AbsClass n)먼저 만들자

		// test라는 메서드를 사용하기 위해 test를 가지고 있는
		// 클래스를 생성한다.
		AbsMain m = new AbsMain();
		m.test(ac1);
		System.out.println(ac1.getValue());//왜 120일까?
	}
```

```
	public void test(AbsClass n){
	   //AbsClass는 부모이므로 자신을 포함한 자식들까지 인자로 받을 수 있다.
		n.changeValue();
	}
}
```

----------예제 2

마지막 예제! 책에 아주 잘 나와있는게 있어서 그걸 한번 해볼게요.
스타크래프트 다들 아시는지 모르겠네요. 전 잘 모릅니다.
근데 보면 세 개의 종족에 지상, 공중유닛이 분리되어 있고.... 유닛마다 공격력이나 방어력이 모두 다르잖아요???

모든 유닛은 상대방에게 '공격을 당한다'라는 액션을 가지고 있지만, 공격당할 때 감소하는 에너지의 수치는 차이가 있을 겁니다.

이렇게 공통적인 기능이 있으나, 내부에서 구현되어야 하는 코드가 다를 때 사용할 수 있는 아주 괜찮은 예제인 것 같습니다.

Unit클래스 정의

```
public abstract class Unit {
	String name;//이름
	int energy;//체력

	//유닛이 공격을 당했을 때 체력 감소량을 관리하기 위한 메서드
	//유닛마다 체력 감소량이 다르기 때문에 추상메서드로 정의했다.
	abstract public void decEnergy();

	public int getEnergy() {
		return energy;
	}
}
```

Terran클래스 정의

```
public class Terran extends Unit{

	//공중유닛 이면 true, 지상유닛이면 false
	boolean fly;

	public Terran(String name, int energy, boolean fly) {
```

```
                super.name = name;
                super.energy = energy;
                this.fly = fly;
        }//생성자 오버로딩.


        @Override
        public void decEnergy() {
                energy -= 3;
        }
}
```

Protoss클래스 정의

```
public class Protoss extends Unit{

        boolean fly;

        public Protoss(String name, int energy, boolean fly) {
                super.name = name;
                super.energy = energy;
                this.fly = fly;
        }

        @Override
        public void decEnergy() {
                energy--;
        }
}
```

Zerg클래스 정의

```
public class Zerg extends Unit{

        boolean fly;

        public Zerg(String name, int energy, boolean fly) {
                super.name = name;
                super.energy = energy;
                this.fly = fly;
        }//생성자 오버로딩.
```

```
	@Override
	public void decEnergy() {
		energy -= 10;
	}
}
```

UnitMain클래스 정의

```
public class UnitMain {
	public static void main(String[] args) {
		Terran t1 = new Terran("해병", 100, false);
		t1.decEnergy();//재정의한 decEnergy()호출
		System.out.println("t1의 energy : " + t1.getEnergy());

		Zerg z1 = new Zerg("무리군주", 200, true);
		z1.decEnergy();
		System.out.println("z1의 energy : " + z1.getEnergy());

		Protoss p1 = new Protoss("거신", 250, false);
		p1.decEnergy();
		System.out.println("p1의 energy : " + p1.getEnergy());
	}
}
```

---예제3

인터페이스

인터페이스는 앞에서 배운 추상클래스와 매우 유사하지만, 서비스 요청에 따른 중계자 역할을 하는 것과 같다.

중국집에 갔다고 치자. 메뉴판을 보고 깐풍기를 골랐는데,
메뉴에 대고 “깐풍기 내놔!”라고 해봤자. 메뉴판이 “옙!!!” 하면서 ‘지잉~’열리며 마치 영화처럼 깐풍기를 대령하지 않는다.

메뉴를 보고 골라서 직원에게 주문을 하면 주방에서 요리를 만들어서 고객에게 제공하는 것처럼 인터페이스는 고객이 호출할 수 있는 서비스의 목록이라고 할 수 있다.

메뉴판에는 분명 짜장면이 있는데 시켜놓고 보니 주방에 밀가루가 없네요??
알고 보니까 짜장면을 만들지 못하는 식당이었다. 라는 상황이 발생하면 안되기 때문에, 식당(인터페이스를 구현할 클래스)에는 메뉴판(인터페이스)에 있는 서비스가 하나도 빠짐없이 구비되어 있어야 한다. 예제로 확인하시죠.

인터페이스의 구성

[접근제한] interface 인터페이스명{
    상수;
    추상메서드;
    //인터페이스에는 상수와 추상메서드 이외에는 아무것도 들어갈 수 없다.
}

프로젝트에서 마우스 우측클릭, new -> interface를 통해 인터페이스 생성

InterTest 인터페이스 정의

```
public interface InterTest {
        final int A = 100;
        abstract int getA();
}
```

InterTestEx 클래스 정의

```
public class InterTestEx implements InterTest{
        //인터페이스를 구현하려면
        //구현하려는 클래스에서 implements예약어를 사용한다.

        @Override
        public int getA() {
                return A; //InterTest의 상수 A를 반환
        }
}
```

InterMain 클래스 정의

```
public class InterMain {

        public static void main(String[] args) {

                InterTestEx it = new InterTestEx();
                System.out.println("getA() : " + it.getA());
        }
}
```

----------예제 1

인터페이스간의 상속

바로 예제로 확인

Inter_Menu1 인터페이스 정의

```
public interface Inter_Menu1 {
```

```
        abstract String jajang();

        //abstract는 생략되어도 interface안에서는 자동으로 추상으로 인식.
        String jjambbong();
}
```

Inter_Menu2 인터페이스 정의

```
public interface Inter_Menu2 {
        abstract String tangsuyuck();//탕수육 철자 모르겠다;;ㅋ
}
```

Inter_Menu3 인터페이스 정의

```
public interface Inter_Menu3 extends Inter_Menu1, Inter_Menu2{
        //인터페이스는 구현능력이 없기 때문에 다중상속이 가능하다
        abstract String boggembab();//보끔밥....
}
```

InterMain 클래스 정의

```
public class InterMain implements Inter_Menu3{
        public static void main(String[] args) {

                InterMain im = new InterMain();

                System.out.println("--우리집 메뉴판--");

                Inter_Menu1 im1 = im;
                Inter_Menu2 im2 = im;
                Inter_Menu3 im3 = im;
                //InterMain의 객체인 im을 Inter_Menu에 대입
                //오류가 날 것 같지만 그렇지가 않다.
                //im이 구현하고 있는 Inter_Menu3이라는 인터페이스가
                //Inter_Menu1과 Inter_Menu2에게서 상속받은 것이므로.

                //하지만 사용 범위가 변환된 각 인터페이스 내에 정의된 메서드들로
                //국한됨을 기억하자.

                System.out.println(im.jajang());

                //im1객체에서는 jjambbong()과 jajang()만 호출할 수 있다.
                System.out.println(im1.jjambbong());
```

```
		//im2객체에서는 tangsuyuck()만 호출할 수 있다.
		System.out.println(im2.tangsuyuck());

		//im3객체에서는  boggembab(), jjambbong(),
		//		jajang(), tangsuyuck()전부 호출 가능
		System.out.println(im3.boggembab());
	}

	@Override
	public String jajang() {
		// TODO Auto-generated method stub
		return "하나죽으면 아는 자장면";
	}

	@Override
	public String jjambbong() {
		// TODO Auto-generated method stub
		return "왜 짬뽕은 잠봉이 아님??";
	}

	@Override
	public String tangsuyuck() {
		// TODO Auto-generated method stub
		return "쌀탕수육";
	}

	@Override
	public String boggembab() {
		// TODO Auto-generated method stub
		return "다섞어 볶음밥";
	}
}
```

예제1

열거형(enum)

열거형은 상수를 가지고 생성되는 객체들을 한곳에 모아둔 하나의 묶음이다.
변수를 사용자가 지정한 이름으로 0부터 순차적으로 증가시켜준다.

EnumMain클래스 정의

```
public class EnumMain{
```

```
	public enum Item{
		Start, Pause, Exit
	}

	public static void main(String[] args) {

		while(true){
			System.out.println("0 : 게임시작");
			System.out.println("1 : 일시정지");
			System.out.println("2 : 게임종료");

			Scanner scan = new Scanner(System.in);
			int n = scan.nextInt();

			Item start = Item.Start;
			Item exit = Item.Exit;
			Item pause = Item.Pause;

			if(n == start.ordinal())
				System.out.println("게임이 시작됨");

			else if(n == pause.ordinal())
				System.out.println("게임이 일시정지됨");

			else{
				System.out.println("게임종료");
				return;
			}
		}
	}
}
```

내부클래스

내부 클래스란 특정 클래스 안에 또 다른 클래스가 정의되는 것을 의미한다.
내부클래스는 독립적이지만 하나의 멤버처럼 사용할 수 있는 특징이 있다.

내부클래스의 종류

Member, Local, Static, Anonymous(익명)

1. Member내부클래스

멤버 내부 클래스는 기존에 외부에서 생성해오던 클래스와 같은 형식이지만
특정 클래스의 내부에서 정의된다는 차이점이 있다.

MemberInner클래스 정의

```
public class MemberInner {

	int a = 10;
	private int b = 100;
	static int c = 200;

	//멤버 내부클래스 정의
	class Inner{
		public void printData(){
			System.out.println("int a : " + a);
			System.out.println("private int b: " + b);
			System.out.println("static int c : " + c);
		}
	}

	public static void main(String[] args) {
		MemberInner outer = new MemberInner(); //Main클래스 객체 생성

		//메인클래스(MemberInner)의 내부클래스(Inner)형 객체 inner는
		//메인클래스 객체(outer)를 통해 내부클래스 객체화(new Inner())
		MemberInner.Inner inner = outer.new Inner();
	  //MemberInner.Inner inner = new MemberInner().new Inner(); //한줄로

		inner.printData();
	}
}
```

2. Local 내부클래스

로컬 내부 클래스는 특정 메서드 안에서 정의되는 클래스를 말한다.
특정 메서드 안에서 선언되는 지역변수와 같은 것이다.

LocalInner클래스 정의

```
public class LocalInner {

	int a = 100;

	//innerTest함수 정의
	public void innerTest(int n){
		int b = 200;//지역변수
```

```
                final int c = n;//상수

                //innerTest()함수 내부에 지역내부클래스 작성
                class Inner{
                        //Local내부 클래스는 외부 클래스(LocalInner)의 멤버변수와
                        //상수들만 접근할 수 있다.

                        public void getData(){
                                System.out.println("int a : " + a);
                                //System.out.println("int b : " + b);오류
                                System.out.println("final int c : " + c);
                        }
                }//내부 클래스의 끝

                //지역내부클래스라고 해서 직접 사용할 수 있는 것은 아니고
                //일반 클래스처럼 객체생성이 반드시 필요하다
                Inner i = new Inner();
                i.getData();
        }
        public static void main(String[] args) {
                LocalInner outer = new LocalInner();//메인클래스 객체 생성

            outer.innerTest(500);//outer에 정의되어 있는 함수(내부클래스가 아니다!)호출
        }
}
```

3. Static 내부클래스

내부 클래스가 static변수를 가지고 있다면, 해당 내부 클래스를 static으로 선언해야 한다.

StaticInner클래스 정의

```
public class StaticInner {

        int a = 10;
        private int b = 100;
        static int c = 200;

        static class Inner{
                //내부클래스에 static변수가 있다면
                //내부 클래스를 static으로 정의해야 한다.
                static int d = 1000;
```

```
		public void getData(){
			//System.out.println("int a : " + a); 오류
			//System.out.println("int b : " + b); 오류

			//↑↑↑↑변수 a 와 b는 static이 아니기 때문에
			//static클래스 내부에서 사용될 수 없다.

			System.out.println("static int c : " + c);
		}
	}
	public static void main(String[] args) {
		StaticInner.Inner inner = new StaticInner.Inner();
		inner.getData();

		//스테틱 클래스는 외부 클래스의 접근 없이 바로 객체생성이 가능
		//Inner inner = new Inner();
		//inner.getData();
	}
}
```

4. 익명 내부클래스

이름이 없는 클래스다.
한번만 사용하고 버려지는 객체를 사용할 때 유용하게 쓰이는 내부 클래스다.
나중에 배울 이벤트 Listener등에서 사용되는 방식이다.
직접 만들어서 쓰는 경우는 많지 않고, 대부분 정의된 것을 지금 해볼 예제처럼 호출해서 사용한다.

4. AnonyInner클래스 정의

```
//인터페이스 먼저 정의
interface TestInter{
	int data = 10000;//final을 쓰지 않아도 자동으로 상수화 됨
	abstract public void printData();
}

//AnonyInner클래스 정의
public class AnonyInner {

	//test()함수 정의
	public void test(){
```

```
		//익명클래스. 인터페이스를 생성한 것이기 때문에 오버라이딩이 필수다
		new TestInter(){

			@Override
			public void printData() {
				System.out.println("익명클래스 printData()");

			}//추상메서드 재정의

		}.printData();//마지막에 재정의한 printData()함수를 호출해줘야 한다.
	}

	public static void main(String[] args) {

		//메인클래스 객체를 생성하고
		//그 객체를 통해 test()함수를 호출한다.
		AnonyInner in = new AnonyInner();
		in.test();
	}
}
```

JFrame과 익명클래스를 이용한 이벤트 예제

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

		Frame t = new Frame();
		Button btn = new Button("종료");

		t.setSize(300, 300);
		t.add(btn);
		t.setVisible(true);

		btn.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent arg0) {

				System.exit(0);
			}
		}
```

```
                });
        }
}
```

Try - Catch(예외처리)

자바에서 프로그램이 실행하는 도중에 예외(에러)가 발생하면 그 시점에서 프로그램이 강제적으로 종료된다. 이것은 올바른 판단이지만, 때로는 예상할 수 있는 가벼운 오류가 있거나, 예외가 발생했을때도 프로그램을 종료하지 않고 이후의 작업을 진행해야 할 때가 있다.
예외처리를 통해 프로그램의 비정상적인 종료를 줄이고 정상적으로 프로그램이 계속 진행될 수 있도록 할 수 있다.

TryCatch 메인클래스 정의

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

                int n = 0;
                int result = 0;

                //try {
                        result = 10 / n;
                        //정수를 0으로 나누면 에러가 난다.
                //} catch (Exception e) {
                        //e.printStackTrace();
                        //System.out.println("오류발생");
                //}

                System.out.println(result);
                //결과 확인 후, 주석처리한 try-catch문을 활성화 시켜서
                //오류가 나지 않는다는 걸 확인시켜줍시다.
        }
}
```

---예제1

TryCatchEx2 클래스 작성

```
public class TryCatchEx2 {
        public static void main(String[] args) {

                int[] var = {10, 20, 30};
                //try {
                        for(int i = 0; i <= var.length; i++)
                                System.out.println("var[" + i + "] = " + var[i]);
```

```
                //} catch (Exception e) {
                        //e.printStackTrace();
                        //System.out.println("오류발생");
                //}
                System.out.println("프로그램 끝");
        }
}
```

----------예제2

문제:
Scanner객체를 통해 정수를 입력받도록 하고
정수를 입력받았을 경우(예외가 발생하지 않은 경우)엔 입력받은 수를 그대로 출력하고
키보드에서 정수 이외의 값을 입력하면
“정수만 입력할 수 있습니다”라는 오류메시지가 출력되도록 하기

```
public class TryCatchEx3 {
        public static void main(String[] args) {

                System.out.print("정수입력 : ");

                Scanner scan = new Scanner(System.in);

                try {
                        int n = scan.nextInt();
                        System.out.println("입력받은 수 : " + n);
                } catch (Exception e) {
                        System.out.println("정수만 입력할 수 있습니다.");
                }
        }
}
```

----------예제3

문제:
Scanner객체를 통해 정수를 입력받도록 하고
입력받은 값이 정수인지 정수가 아닌지를 판별하여 아래와같이 출력하도록 하자.

결과:

```
//정수를 입력 받은 경우
정수입력 : 100
결과 : 100
```

```
//정수를 입력 받지 않은 경우
```
정수입력 : aaa
aaa은(는) 정수가 아닙니다.

풀이 :

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

		System.out.print("정수입력 : ");
		Scanner scan = new Scanner(System.in);
		String str = "";

		try {

			str = scan.next();
			System.out.println("결과 : " + Integer.parseInt(str));

		} catch (Exception e) {
			System.out.println(str + "은(는) 정수가 아닙니다.");
		}
	}
}
```

----------예제

스레드 - page 636

스레드는 독립적인 실행단위입니다.
우리가 한글 문서를 작성 하면서 프린트를 사용해서 인쇄하는걸 동시에 할 수 있는 것.
혹은 인터넷을 하면서 음악을 듣는 것처럼 한번에 두가지 이상의 프로세스(운영체제에서 실행 중인 하나의 프로그램)를 실행 가능하게 해 준다.
실제로 동시에 두 개가 실행되는 것은 아니고, 운영체제 내부에서 동시에 돌아가는 것처럼 보이도록 아주 빠르게 번갈아서 스레드를 실행하는 것.

싱글스레드.

ThreadEx 클래스 정의

```
public class ThreadEx extends Thread{
```

```
	@Override
	public void run() {
		//프로세스의 독립적인 수행을 위한 영역
		for (int i = 0; i < 10; i++) {
			System.out.println("스레드 실행");
		}
	}
}
```

ThreadMain클래스 정의

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

		ThreadEx t = new ThreadEx();
		t.start();//run()메서드를 호출하는 메서드
		//t.run();은 run()메서드를 독립적으로 수행하지 못한다.
		//즉, 일반 메서드처럼 호출해버림.
		//run()메서드의 내용을 별개로 수행하고 싶다면
		//t.start()를 이용해야 함.
		System.out.println("main종료");
	}
}
```

멀티스레드.

Thread_test1클래스 정의

```
public class Thread_test1 extends Thread{

	@Override
	public void run() {//스레드 클래스에는 스레드 작동을 위한 run()메서드가 있다.

		for(int i = 0; i < 50; i++){

			System.out.print("1");
		}
	}
}
```

Thread_test2클래스 정의

```
public class Thread_test2 extends Thread{
```

```
	@Override
	public void run() {
		for(int i = 0; i < 50; i++){

			System.out.print("2");
		}
	}
}
```

Thread_Ex클래스 정의

```
public class Thread_Ex_Main{
	public static void main(String[] args) {
		Thread_test1 t1 = new Thread_test1();
		Thread_test2 t2 = new Thread_test2();

		t1.start();
		t2.start();
		//실행하면 스레드는 동시에 진행되는 것이 아니라
		//번갈아가며 실행된다는 것을 알 수 있다.
	}
}
```

----------예제 1

Thread_Ex클래스 정의(스레드 sleep예제)

```
public class Thread_Ex extends Thread{

	private int[] temp;

	public Thread_Ex(){//생성자
		temp = new int[10];
		for(int i = 0; i < temp.length; i++){
			temp[i] = i;
		}
	}

	@Override
	public void run() {//스레드 클래스에는 스레드 작동을 위한 run()메서드가 있다.

		for(int i : temp){
```

```
                        try {
                                Thread.sleep(1000);//1초 대기
                        } catch (Exception e) {
                                e.printStackTrace();
                        }

                        System.out.println("temp : " + temp[i]);
                }
        }
}
```

Thread_Ex_Main클래스 정의

```
public class Thread_Ex_Main {
        public static void main(String[] args) {
                Thread_Ex t1 = new Thread_Ex();
                t1.start(); //start()메서드를 호출하면 run()메서드 내부의 연산을 처리.

                try{
                        Thread.sleep(2000);
                        System.out.println("프로그램 종료");
                }catch(Exption e){
                        e.printStackTrace();
                }
        }
}
```

----------예제2

Runnable인터페이스를 이용한 스레드 생성법

Thread_test클래스 정의

```
public class Thread_test1 implements Runnable {

        int temp[];
        public Thread_test1(){
                temp = new int[10];
                for(int i = 0; i < temp.length; i++){
                        temp[i] = i;
                }
        }

        @Override
```

```
	public void run() {
		for(int i = 0; i < temp.length; i++){

			try {
				Thread.sleep(1000);
			} catch (Exception e) {
				e.printStackTrace();
			}
			System.out.println("temp: " + temp[i]);
		}
	}
}
```

Thread_Ex_Main클래스 정의

```
public class Thread_Ex_Main{
	public static void main(String[] args) {
		//Runnable을 구현하는 클래스를 이용한 객체화
		Thread_test1 t1 = new Thread_test1();

		//스레드를 생성하여 파라미터로 Runnable객체를 넣어줘야 한다.
		Thread t2 = new Thread(t1);
		t2.start();
	}
}
```

예제3

Thread_Ex_Main클래스 정의

```
public class Thread_Ex_Main implements Runnable{
	public static void main(String[] args) {

		System.out.println("메인클래스 시작");
		Thread_Ex_Main t1 = new Thread_Ex_Main();
		//Runnable t1 = new Thread_Ex_Main();
		Thread mThread = new Thread(t1);//스레드 생성
		mThread.start();//run()호출
		System.out.println("메인클래스 종료");

		//결과를 보면
		//메인 클래스 시작
		//run()
		//first()
```

```
		//second()
		//메인클래스 종료
		//위의 순서대로 나와야 할 것 같지만 실제로는 그렇지 않다.

		//메인클래스 시작
		//메인클래스 종료
		//run()
		//first()
		//second()
		//이렇게 출력이 된다.

		//메인클래스가 실행된 이후에
		//스레드 객체를 호출하고
		//start()를 사용해 run()을 호출했지만
		//run()이 호출되기 전에
		//그 아래줄인 '메인클래스 종료'가 먼저 출력되었기 때문

		//이처럼 스레드를 사용하면
		//위에서 아래로, 좌에서 우로 라고 하는
		//작업순서를 따르게 되지 않을수도 있다.
	}

	@Override
	public void run() {
		// TODO Auto-generated method stub
		System.out.println("run()");
		first();
	}

	public void first(){
		System.out.println("first()");
		second();
	}

	public void second(){
		System.out.println("second()");
	}
}
```

예제4

데몬쓰레드

데몬 쓰레드는 다른 일반 쓰레드의 작업을 돕는 보조적인 역할을 수행하는 쓰레드이다.
함께 구동중인 일반 스레드가 종료되면 데몬스레드도 함께 종료된다.
예를들어 문서를 작성하는 도중에 3초 간격으로 자동 세이브가 필요하다고 가정하여 코드를 작성해 보자.

DaemonMain 클래스 생성

```
public class DaemonMain implements Runnable{

	static boolean autoSave = false;

	public static void main(String[] args) {

		DaemonMain dm = new DaemonMain();
		Thread t = new Thread(dm);

		//t라는 스레드가 데몬스레드임을 명시하는 메서드.
		//메인스레드가 종료될 때 함께 종료되도록 한다.
		t.setDaemon(true);

		//run()메서드 시작
		t.start();

		for(int i = 1; i <= 15; i++){

			try {
				Thread.sleep(1000);

			} catch (Exception e) {
				// TODO: handle exception
			}

			System.out.println(i);

			if(i == 3)//3초 뒤에 자동세이브 시작
				autoSave = true;
		}

		System.out.println("프로그램 종료");
	}

	@Override
```

```
	public void run() {

		while(true){

			try {
				Thread.sleep(3000);

			} catch (Exception e) {
				// TODO: handle exception
			}

			if(autoSave == true)
				System.out.println("자동저장을 수행합니다.");

		}
	}//run()
}
```

스레드의 join기능
이전 예제와 같지만, join메서드를 이용하여, 작업의 결과물을 다르게 출력해보자.

ThreadJoin 클래스 정의

```
public class ThreadJoin implements Runnable{
	public static void main(String[] args) {

		System.out.println("메인클래스 시작");
		ThreadJoin t1 = new ThreadJoin();
		//Runnable t1 = new ThreadJoin();
		Thread mThread = new Thread(t1);//스레드 생성
		mThread.start();//run()호출

		try {
			//join메서드를 사용하면
			//해당스레드의 run()메서드가 완전히 작업을 마칠때까지
			//메인스레드에서 응답을 기다리게 된다.
			mThread.join();
		} catch (Exception e) {
			// TODO: handle exception
		}
		System.out.println("메인클래스 종료");
	}
```

```
	@Override
	public void run() {
		// TODO Auto-generated method stub
		System.out.println("run()");
		first();
	}

	public void first(){
		System.out.println("first()");
		second();
	}

	public void second(){
		System.out.println("second()");
	}
}
```

예제5

문제 :
스캐너를 이용하여 키보드에서 숫자를 입력받고
스레드에서 입력받은 숫자가 1씩 감소하다가 0이 되었을 때
“종료”라는 메시지와 함께 스레드를 빠져나오도록 만들어보자.

```
public class ThreadCount implements Runnable{

	private int n;

	public ThreadCount(int n) {
		this.n = n;
	}

	public static void main(String[] args) {

		System.out.print("값을 입력 : ");
		Scanner scan = new Scanner(System.in);

		ThreadCount t = new ThreadCount(scan.nextInt());
		Thread tt = new Thread(t);
		tt.start();
	}
```

```
	public void run() {
		for(int i = n; i >= 0; i--){

			try {
				System.out.println(i);
				Thread.sleep(1000);

			} catch (Exception e) {
				// TODO: handle exception
			}
		}
		System.out.println("종료");
	}
}
```

----------예제(문제)6

(심심풀이) Thread.sleep()을 이용하여
희한한 달리기 게임을 만들어보자ㅎㅎ(함께 해보기)

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

		int[] playerRandom = new int[4];
		String[] playerJump = {"", "", "", ""};

		boolean finish = false;
		int finishPlayer = 0;

		while (true) {

			//도약거리
			for(int i = 0; i < playerRandom.length; i++){
				playerRandom[i] = new Random().nextInt(3);
			}

			//0.1초 간격으로 휴식
			try {
```

```
                Thread.sleep(100);

            } catch (Exception e) {
                // TODO: handle exception
            }

            //각 선수에게 도약거리 적용
            for(int i = 0; i < playerJump.length; i++){

                switch (playerRandom[i]) {
                case 0:
                    playerJump[i] += "";
                    break;

                case 1:
                    playerJump[i] += " ";
                    break;

                case 2:
                    playerJump[i] += "  ";
                    break;
                }
            }//for

            //달리기
System.out.println("-----------------------------------------");
            for(int i = 0; i < playerJump.length; i++){

                System.out.print(playerJump[i]);
                System.out.println(i + 1 + "옷");

                if(playerJump[i].length() >= 50){
                    finishPlayer = i + 1;
                    finish = true;
```

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

                //결산
                if(finish){
                    System.out.println("winner - "
                                        + finishPlayer);
                    break;
                }
            }//while
        }//main end
}//class end
```

----------달리기

자바문제 -스레드(암산퀴즈) 출제!!

스레드 동기화(Synchronized)

휴대폰으로 음악이나 영상을 동기화 하게되면, 동기화가 끝날때까지 휴대폰은 다른작업을 할 수가 없죠?? 그게 이 동기화 스레드의 구조인데요
두 개 이상의 스레드가 하나의 자원을 공유할 경우, 동기화 문제가 발생한다.
변수는 하나인데, 두 개의 스레드가 동시에 한 변수의 값을 변경하다가 오류가 발생할수도 있기 때문입니다.

내가 컴퓨터로 작업을 하다가 잠시 자리를 비운 사이에 누군가가 내 컴퓨터에 손을 대서 내가 하고 있던 작업내용을 바꿔버리지 못하게 하기 위해서, 내 문서작업이 끝날때까지 다른사람이 손대지 못하도록 컴퓨터를 잠궈둘 필요가 있다.

이처럼 특정 스레드들이 공유하는 한 개의 자원을 사용중일 때, 현재 자원을 사용중이지 않은 다른스레드가 작업을 수행하지 못하게 하기 위해 동기화가 필요하다.

SynchronizedEx 클래스 정의

```
public class SynchronizedEx implements Runnable{

    private long money = 10000;//잔액
    //money로 셋터겟터부터 만듦
    //run()정의
```

```
	@Override
	public void run() {

		//synchronized키워드를 사용하면
		//해당 키워드가 명시되어있는 영역이 마무리 될 때까지
		//다른 스레드에서 접근하지 못하게 된다.
		synchronized (SynchronizedEx.class) { //this

			for(int i = 0; i < 10; i++){

				try {
					Thread.sleep(500);
				} catch (Exception e) {
					e.printStackTrace();
				}

				if(getMoney() <= 0)
					break;//잔액이 0이면 for문을 탈출
				//Main까지 만들어 결과 찍어서 보여준 후에
				//if문 주석달고 son이 구동되는 것도 확인해보자.
//그리고 위의 synchronized (SynchronizedEx.class) 영역도 주석처리해서
//엄마와 아들이 동시에 돈을 찾는 현상도 확인해보자

				//outMoney()함수 먼저 정의한 후에 추가
				outMoney(1000);
			}//for
		}//synchronized
	}//run()

	public long getMoney() {
		return money;
	}

	public void setMoney(long money) {
		this.money = money;
	}

	public void outMoney(long howMuch){//출금...ㅠㅠ;; 영어모름ㅠ

		//해당 스레드의 이름을 가져옴.
```

```
                //이름은 해당 스레드를 생성하는 클래스에서 기재하게 됨
                String threadName = Thread.currentThread().getName();

                if(getMoney() > 0){//잔액이 0이상일때 출금가능
                    money -= howMuch;//잔액에서 출금액을 마이너스
                    System.out.println(threadName + " - 잔액 : " + getMoney() + "원");

                }else{
                        System.out.println(threadName + " - 잔액이 없습니다.");
                }
        }
}
```

SyncMain 클래스 정의

```
public class SyncMain {
        public static void main(String[] args) {
                SynchronizedEx atm = new SynchronizedEx();
                Thread mom = new Thread(atm, "엄마");
                //Thread.currentThread().getName();이 "엄마"가 된다.

                Thread son = new Thread(atm, "아들");

                mom.start();
                son.start();
        }
}
```

---예제6

wait()과 notify()

스레드가 진행중에 wait()을 만나면 일시적으로 정지된다.
스레드가 구동되고 있을 때 일시적으로 대기상태에 보내고, 다른 스레드에게 제어권을 넘긴다.
wait()을 만나 대기상태에 빠진 스레드는 notify()를 만나 재 구동된다.
두 개 이상의 스레드가 구동중일 때 한 개의 동기화 스레드가 작업을 완전히 마칠때까지 기다렸다가 다른 스레드의 작업이 수행되는 것이 아니라 동기화 진행중에 일시적으로 스레드를 정지시키고 다른 스레드가 작업을 할 수 있다.

Account클래스 정의

```
public class Account {

	int balance = 1000;//잔액

	//출금메서드
	public synchronized void withdraw(int money){

		//잔액이 출금액보다 적을 경우
		if(balance < money){

			try {

				wait();//스레드가 일시적으로 정지상태에 빠짐

			} catch (Exception e) {
				// TODO: handle exception
			}
		}

		balance -= money;
	}//withdraw()

	//입금메서드
	public synchronized void deposit(int money){
		balance += money;
		notify();//정지된 스레드를 실행
	}
}
```

AccountThread클래스 생성

```
public class AccountThread implements Runnable{

	Account acc;//Account객체 준비

	//생성자 정의
	public AccountThread(Account acc) {
```

```
            this.acc = acc;
        }

        @Override
        public void run() {

            while(true){

                try {

                    Thread.sleep(500);

                } catch (Exception e) {
                    // TODO: handle exception
                }

                //출금액을 100 ~ 300원 사이의 난수로 지정
                int money = (new Random().nextInt(3) + 1) * 100;
                acc.withdraw(money);
                System.out.println("잔액 : " + acc.balance);
            }
        }
}
```

AccountMain클래스 정의

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

            Account acc = new Account();

            Runnable r = new AccountThread(acc);
            Thread t1 = new Thread(r);

            t1.start();//스레드 구동

            //스레드와는 별개로 값을받아 입금 시키는
            //코드를 수행할 while()문
```

```
                while(true){
                        Scanner scan = new Scanner(System.in);
                        int n = scan.nextInt();
                        acc.deposit(n);
                }
        }
}
```