1주차(4)

메서드 오버로딩 - p152

오버로딩은 메서드의 '중복정의' 라고 하며, 하나의 클래스 내에서 같은 이름을 가진 메서드(함수)가 여러개 정의되는 것을 말한다.
메서드명은 대소문자를 비롯해서 반드시 같게 만들어야 하고, 인자들은 인자명을 제외한 인자의 수가 다르든, 인자의 수가 같을 경우 인자의 자료형이 다르든, 다른 메서드에 배치된 인자들과 반드시 다르게 정의되어야 한다.
오버로딩은 다양한 메서드들을 같은 이름으로 작업할 수 있다는 의미. 효율성이 높다.

클래스 작성

```
public class Ex1_Overloading {

	public void getResult(int n){
		System.out.println(n + "은(는) int입니다.");
	}

	public void getResult(char n){
		System.out.println(n + "은(는) char입니다.");
	}

	public void getResult(String str){
		System.out.println(str + "은(는) String입니다.");
	}

	public void getResult(int n, String str){
		System.out.println(n + "은(는) int\n" + str + "은(는) String");
	}
}
```

메인클래스 작성

```
public class Ex1_OverloadingMain {
	public static void main(String[] args) {

		Ex1_Overloading ex = new Ex1_Overloading();//명시적 객체 생성
		ex.getResult('A'); //char
		ex.getResult(10); //int
		ex.getResult("Hi"); //String
		ex.getResult(5, "Hello"); //int, String
	}
}
```

---------- ex1

메서드 오버로딩 문제

Method클래스를 만들어 각기 다른 기능을 하는 makeBread()메서드를 세 개만드는데,
메인클래스에서 각각의 메서드를 호출했을때의 결과를 보고 로직을 구현해 보자.

결과 :

```
빵을 만들었습니다. //첫번째 메서드 호출 결과
--------------------------------
빵을 만들었습니다.
빵을 만들었습니다.
요청하신 2개의 빵을 완성했습니다. //두번째 메서드 호출 결과
--------------------------------
크림빵을 만들었습니다.
크림빵을 만들었습니다.
크림빵을 만들었습니다.
요청하신 3개의 크림빵을 완성했습니다.//세번째 메서드 호출 결과
```

Method클래스 생성. 오버로딩 메서드 정의

```
public class Method {

	void makeBread()
	{
		System.out.println("빵을 만들었습니다.");
	}

	void makeBread(int count)
	{
		for(int i = 0; i < count; i++)
			System.out.println("빵을 만들었습니다.");
		System.out.println("요청하신 " + count + "개의 빵을 완성했습니다.");
	}
```

```
	void makeBread(int count, String name)
	{
		for(int i = 0; i < count; i++)
			System.out.println(name + "을 만들었습니다.");
		System.out.println("요청하신 " + count + "개의 " + name + "을 완성했습니다.");
	}
}
```

오버로딩을 위한 메인클래스 생성

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

		Method mh = new Method();

		mh.makeBread(); //예제1
		System.out.println("------------------------------");

		mh.makeBread(10); //예제2
		System.out.println("------------------------------");

		mh.makeBread(3, "크림빵");//예제3
	}
}
```

생성자 - p156

객체가 생성될 때 메모리 할당 및 멤버변수의 초기화를 목적으로 사용하는 것.
객체가 생성될 때 딱 한번만 호출된다는 것 잊지말긔.
//생성자 개념은 직전에 했던 빵 만들기 클래스와 노트클래스를 새로 만들어서 설명한번 더 해줄 것.

생성자 클래스 작성

```
public class Ex2_Construct {

	//기본적으로는 숨겨져 있어서 보이지 않는다.
	//불러내서 내부를 따로 정의 할 수 있다.
	Ex2_Construct(){
		System.out.println("생성자 호출");
	}
	//이게 생성자. 생성자는 반환형이 없다. 첫번째 예제(두번째 할 때 주석처리)
```

```
	public Ex2_Construct(String name) {
		System.out.println("name이 " + name + "으로 초기화 됨");
	}
	//두번째 예제(세번째 할 때 주석처리). 생성자에 파라미터를 넣어서 정의할 수 있음

	public Ex2_Construct() {
		n = 100; //생성자에서 n을 100으로 초기화
	}
	//세번째 예제.

	int n;

	public int getN() {
		return n;
	}

	public void setN(int n) {
		this.n = n;
	}
}
```

메인클래스 정의

```
public class Ex2_ConstructMain {
	public static void main(String[] args) {
		Ex2_Construct con = new Ex2_Construct(); //예제 1(끝나고 주석)
		//난 객체 생성만 함!! 그러나 실행하면!!!!!!!!
		//생성자 호출됨

		//이제 생성자에 인자 넣어서 해보자
		//Ex2_Construct con2 = new Ex2_Construct(); //이거 오류나는 이유 설명후 주석
		Ex2_Construct con2 = new Ex2_Construct("염소똥");//예제2(끝나고주석)

		//생성자 초기화 예제
		Ex2_Construct con3 = new Ex2_Construct();
		System.out.println(con3.getN());//100으로 초기화되어 있다.

		con3.setN(200);
		System.out.println(con3.getN());//setter로 값을 수정할 수 있다.
	}
}
```

생성자 오버로딩 - p159

생성자 오버로딩 클래스 정의

```
public class Ex3_ConstructOverloading {

	public Ex3_ConstructOverloading() {
		System.out.println("기본 생성자");
	}

	public Ex3_ConstructOverloading(int n){
		System.out.println("int 자료형을 받는 생성자");
	}//인자의 자료형이나 인자의 갯수가 달라야 오류가 나지 않는다.
	//일반 메서드 오버로딩과 동일

	public Ex3_ConstructOverloading(int n, char c){
		System.out.println("int, char 자료형을 받는 생성자");
	}
}
```

생성자 오버로딩 메인 클래스 정의

```
public class Ex3_ConstructOverloadingMain {
	public static void main(String[] args) {
		Ex3_ConstructOverloading ex = new Ex3_ConstructOverloading();
		Ex3_ConstructOverloading ex2 = new Ex3_ConstructOverloading(10);
		Ex3_ConstructOverloading ex3 =
					new Ex3_ConstructOverloading(10, 'A');
	}
}
```

---------- 예제 1

생성자 오버로딩 클래스2 정의

```
public class Ex4_ConTest {

	private int age;
	private String name;

	public Ex4_ConTest(String name) {
		this.name = name;//this는변수 자신을 포함하고있는 최상위 클래스(객체)
	}
```

```
	public Ex4_ConTest(int age, String name) {
		this.age = age;
		this.name = name;
	}

	public Ex4_ConTest(String name, int age) {
		this.age = age;
		this.name = name;
	}

	public String getResult(){
		String result;
		result = "이름 : " + name + ", 나이 : " + age;
		return result;
	}
}
```

생성자 오버로딩 클래스2 메인 정의

```
public class Ex4_ConTestMain {

	public static void main(String[] args) {
		Ex4_ConTest ex1 = new Ex4_ConTest("김ㅋㅋ");
		Ex4_ConTest ex2 = new Ex4_ConTest(20, "박ㅋㅋ");
		Ex4_ConTest ex3 = new Ex4_ConTest("윤ㅋㅋ", 30);

		System.out.println(ex1.getResult());
		System.out.println(ex2.getResult());
		System.out.println(ex3.getResult());
	}
}
```

static변수 - p167

변수가 static으로 선언되면 객체를 생성하지 않고도 사용할 수 있다.
그리고 현재 static변수를 가지는 객체를 아무리 많이 생성한다고 해도 static변수는 오직 하나만 만들어 진다.
하나만 만들어 진다는 것의 의미는 은행의 이자율을 예로 들어주면 될 듯.

```
public class Ex5_StaticTest {

	int n; //일반적인 멤버변수 선언
```

```
	static String str;

	public static void main(String[] args) {

		str = "안녕하세요";
		Ex5_StaticTest ex1 = new Ex5_StaticTest();

		//n = 1000;
		//오류. n이 선언된 곳과 n = 1000을 정의하는 공간이 다르기 때문에 오류
		ex1.n = 1000; //ex1객체에 접근해서 해결.

		System.out.println(str);
	}
}
```

Static메서드 - p168

static변수와 동일한 개념이지만, static메서드 안에서는 static이 아닌 일반 멤버변수를 참조할 수 없다.

스테틱함수 클래스 정의

```
public class Ex6_StaticFunction {
	String str1 = "일반 멤버변수";
	static String str2 = "Static 변수";

	public static String getString(){
		//return str1; //오류. 스테틱 내부에서는 static변수의 접근만 가능
		return str2;
	}
}

public class Ex6_StaticFunctionMain {
	public static void main(String[] args) {

		String str;
		str = Ex6_StaticFunction.getString();
		//스테틱 메서드는
		//클래스명.메서드 로 접근 가능하다.

		System.out.println("str = " + str);
	}
}
```

----------예제 1

static변수를 사용해서 은행 이자율을 관리해보자.

Back클래스 정의

```
public class Ex7_Bank {

	private String point;//은행 위치
	private String tel;//은행 전번
	static float interest;//은행이자

	//생성자 써먹어보자
	public Ex7_Bank(String point, String tel) {
		this.point = point;
		this.tel = tel;
	}

	//결과를 출력할 메서드
	public void getBank(){
		System.out.println("지점 : " + point + "\n" +
				"전화번호 : " + tel + "\n" +
				"이자율 : " + interest);
	}
}
```

BackMain 클래스 정의

```
public class Ex7_BankMain {
	public static void main(String[] args) {

		Ex7_Bank bk1 = new Ex7_Bank("강남", "02-111-2222");
		Ex7_Bank.interest = 0.2f;
		bk1.getBank();

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

		Ex7_Bank bk2 = new Ex7_Bank("분당", "031-333-4444");
		//Ex7_Bank.interest = 0.2f;
		//이걸 하지 않아도 위에서 interest에 값을 넣었기 때문에
		//bk2에도 적용된다네

		bk2.getBank();
```

```
        }
}
```

자바 강의 1주차(4) 문제(스테틱 변수를 활용한 여행가이드) 풀기!!!