클릭하세요 자바

3부 클래스와 객체

– 13장 생성자

최문환

13장 생성자

1. 생성자의 정의와 호출
2. 레퍼런스 this
3. 생성자 this( )

1. 생성자의 정의와 호출

(1)생성자는 인스턴스가 생성될 때 호출되는 인스턴스 초기화 메서드
(2) 객체의 멤버변수(속성:인스턴스 변수)를 초기화
(3) 생성자 호출:new 클래스명(); 에 의해서 호출
(4) 인스턴스(객체) 생성시에 실행되어야 할 작업을 위해서도 사용
(5) 생성자도 오버로딩이 가능하다.
(6) 매개변수가 없는 생성자를 기본생성자라 한다. 기본생성자는 자바 컴파일러가 기본으로 제공하기 때문에 묵시적으로 생략가능하다.
(7) 하지만 생성자가 오버로딩 되면 자바는 더 이상 기본생성자를 묵시적으로 제공하지 않는다. 그러므로 생성자 호출문제 에러가 발생할 수 있기 때문에 명시적인 기본생성자를 정의하는 것이 좋다.

1.1 생성자 정의하기

1. 생성자는 메서드의 일종이지만 생성자의 이름은 반드시 클래스 이름과 동일해야 한다.
2. 생성자는 리턴형을 기술하지 않는다. 함수의 자료형이 없다.

★ 생성자의 기본 형식

```
접근_지정자    클래스_이름(인수1, 인수2,...){
    문장1;
}
```

<예제> 생성자 정의하기

```
001:class MyDate{
002:  private int year;
003:  private int month;
004:  private int day;
005:  public  MyDate(){
006:    System.out.println("[생성자] : 객체가 생성될 때 자동 호출됩니다.");
007:  }
008:  public void print(){
009:  System.out.println(year+ "/" +month+ "/" +day);
010:  }
011:
012:}
013:public class ConstructorTest02 {
014:  public static void main(String[] args) {
015:    MyDate d =  new MyDate();
016:    d.print();
017:  }
018:}
```

<예제> 생성자의 정의와 호출

```
001:class MyDate{
002:  private int year;
003:  private int month;
004:  private int day;
005:  public MyDate(){
006:    year=2023;
007:    month=4;
008:    day=1;
009:  }
010:  public void print(){
011:  System.out.println(year+ "/" +month+ "/" +day);
012:  }
013:}
014:public class ConstructorTest03 {
015:  public static void main(String[] args) {
016:    MyDate d=new MyDate();
017:    d.print();
018:  }
019:}
```


1.2 생성자 오버로딩

<예제> -전달인자를 갖는 생성자 정의

```
001:class MyDate{
002:  private int year;
003:  private int month;
004:  private int day;
005:  //생성자는 속성(멤버변수)들의 초기화 작업을 목적으로 한다.
006:
007:  //[1] 전달인자 없는 생성자 정의
008:  public MyDate(){
009:    year=2023;    month=4;    day=1;
010:  }
011:  //[2] 전달인자 있는 생성자 정의
012:  public MyDate(int new_year, int new_month, int new_day){
013:    year=new_year;
014:    month=new_month;
015:    day=new_day;
016:  }
```

1.2 생성자 오버로딩

<예제> -전달인자를 갖는 생성자 정의

```
017:
018:  public void print(){
019:  System.out.println(year+ "/" +month+ "/" +day);
020:  }
021:}
022:
023:public class ConstructorTest04 {
 024:  public static void main(String[] args) {
025:    MyDate d=new MyDate();
026:    d.print();
027:
028:    MyDate d2=new MyDate(2022, 7, 19);
029:    d2.print();
030:  }
031:}
```

1.3 디폴트 생성자

<예제> 자바가 제공해주는 디폴트 생성자 호출

```
001:class MyDate{
002:  private int year;
003:  private int month;
004:  private int day;
005:  public void print(){
006:     System.out.println(year+ "/" +month+ "/" +day);
007:  }
008:}
009:public class ConstructorTest01 {
010:  public static void main(String[] args) {
011:    MyDate d=new MyDate();   //디폴트 생성자 호출
012:    d.print();
013:  }
014:}
```

<예제>디폴트 생성자 부재로 인한 에러 발생

```
001:class MyDate{
002:  private int year;
003:  private int month;
004:  private int day;
005:  public MyDate(int new_year, int new_month, int new_day){
006:    year=new_year;
007:    month=new_month;
008:    day=new_day;
009:  }
010:  public void print(){
011:    System.out.println(year+ "/" +month+ "/" +day);
012:  }
013:}
014:
015:public class ConstructorTest05 {
016:  public static void main(String[] args) {
017:    MyDate d=new MyDate();
018:    d.print();
019:
020:    MyDate d2=new MyDate(2022, 7, 19);
021:    d2.print();
022:  }
023:}
```

2. 레퍼런스 this

1.클래스를 구성하는 인스턴스변수는 객체 생성할 때마다 새롭게 메모리 할당을 하기 때문에 객체 단위로 따로 관리된다.

2.하지만 멤버함수는 모든 객체가 이 멤버함수를 공유해서 사용한다.
3.this 는 참조변수로 인스턴스 자신을 가리킨다.
4.this 를 사용할 수 있는 것은 인스턴스 변수 이다.
static 으로 정의된 클래스 메서드에는 인스턴스변수와 this 를 사용할 수 없다.

2.1 this를 사용해야만 하는 경우

메서드(생성자 포함)의 매개변수와 클래스 멤버변수가 동일한 이름일 경우 전달인자와 속성이 구분되지 않기 때문에 문제가 발생하는데 이를 구분 짓기 위해서 속성 앞에 레퍼런스 this를 덧붙인다.
멤버 변수 앞에 this를 붙여야 값이 저장된다.

<예제> 멤버변수와 매개변수 동일해서 생긴 문제점

```
001:class MyDate{
002:  private int year;
003:  private int month;
004:  private int day;
005:  //생성자 정의하기
006:  public MyDate(){
007:  }
 008:  public MyDate(int new_year, int new_month, int new_day){
009:    year=new_year;    month=new_month;    day=new_day;
010:  }
011:  //매개변수가  클래스 인스턴스 변수 이름과 동일한 메서드
012:  public void SetYear(int year){
013:    //this.year=year;
014:    year=year; //값이 저장 안됨.
015:  }
```

<예제> 멤버변수와 속성이 동일해서 생긴 문제점

```
016:  public void SetMonth(int new_month){
017:   month=new_month;
018: }
019:  public void print(){
020:  System.out.println(year+ "/" +month+ "/" +day);
021:  }
022:}
023:
024:public class ConstructorTest06 {
025:  public static void main(String[] args) {
026:    MyDate d=new MyDate(2022, 7, 19);
027:    d.print();
028:    d.SetYear(2023);   //변경되지 않음
029:    d.print();          //-----------------.
030:    d.SetMonth(8);    //변경됨
031:    d.print();          //-----------------.
032:  }
033:}
```


<예제> 속성(멤버변수) 앞에 레퍼런스 this 붙이기

```
001:class MyDate{
002:  private int year;
003:  private int month;
004:  private int day;
005:  public MyDate(){
006:  }
007:  //생성자 역시 매개변수의 이름을 속성과 동일하게 줄 수 있다.
008:  public MyDate(int year, int month, int day){
009:    //멤버변수로 속성 값을 초기화하려면 대입연산자 왼쪽에 this를 붙여야 한다.
010:    this.year=year;  this.month=month;  this.day=day;
011:  }
012:  public void SetYear(int year){ //대입연산자 왼쪽에 this를 붙였기에
013:    this.year=year;              //속성 값이 변경됨
014:  }
015:  public void SetMonth(int month){//대입연산자 왼쪽에 this를 붙였기에
016:    this.month=month;            //속성 값이 변경됨
017: }
```

<예제> 속성(멤버변수) 앞에 레퍼런스 this 붙이기

```
018: public void print(){
019:  System.out.println(year+ "/" +month+ "/" +day);
020:  }
021:}
022:
023:public class ConstructorTest07 {
024:  public static void main(String[] args) {
025:    MyDate d=new MyDate(2022, 7, 19);
026:    d.print();
027:    d.SetYear(2023);  //2023년으로 변경
028:    d.SetMonth(8);    //8월로 변경
029:    d2.print();
030:  }
031:}
```

3. 생성자 this( )

<예제> this()로 같은 클래스내의 다른 생성자 호출하기

```
001:class MyDate{
002:  private int year;
003:  private int month;
004:  private int day;
005:  public MyDate(){
 006:    this(2023, 1, 1);                  //14:에 정의된 생성자 호출
007:  }
008:  public MyDate(int new_year){
009:    this(new_year, 1, 1);               //14:에 정의된 생성자 호출
010:  }
011:  public MyDate(int new_year, int new_month){
012:    this(new_year, new_month, 1);     //14:에 정의된 생성자 호출
013:  }
014:  public MyDate(int new_year, int new_month, int new_day){
015:    year=new_year;
016:    month=new_month;
017:    day=new_day;
018:  }
```

3. 생성자 this( )

```
019:
020:  public void print(){
021:  System.out.println(year+ "/" +month+ "/" +day);
022:  }
023:}
024:
025:public class ConstructorTest10 {
026:  public static void main(String[] args) {
027:    MyDate d=new MyDate(2022, 7, 19);  //14:에 정의된 생성자 호출
028:    d.print();
029:    MyDate d2=new MyDate(2022, 7);     //11:에 정의된 생성자 호출
030:   d2.print();
031:    MyDate d3=new MyDate(2022);       //8:에 정의된 생성자 호출
032:    d3.print();
033:    MyDate d4=new MyDate();           //5:에 정의된 생성자 호출
034:    d4.print();
035:  }
036:}
```

<문제>

1. Animal 클래스에 생성자를 추가하시오. (Ex13_01.java)

```
class Animal{
  String name;
  int age;
  public  void show( ){
    System.out.println( name + "는(은) " + age + " 살입니다.");
  }
}
public class Ex13_01{
  public static void main(String[] args) {
    Animal a1=new Animal("원숭이", 26);
    a1.show( );
  }
}
```

[결과]
원숭이는(은) 26 살입니다.

<문제>

2. Product 클래스에 생성자를 추가하시오.(Ex13_02.java)

```
class Product{
  String name;
  int price;
}
public class ConEx02 {
  public static void main(String[] args) {
    Product p1=new Product("웰치스", 700);
    Product p2 =new Product("커피");
    Product p3 =new Product(500);
    Product p4 =new Product( );
    System.out.println(p1.name + ", " + p1.price);
    System.out.println(p2.name + ", " + p2.price);
    System.out.println(p3.name + ", " + p3.price);
  }
}
```

[결과]
웰치스, 700
커피, 800
물, 500

<문제>

3. 다음은 Thing 클래스를 설계한 것입니다. new Thing()와 같이 기술해서 성공적으로 생성자를 호출하는 Thing 클래스를 ▪다음▪ 중에서 고르시오.

▪가. class Thing { //그리고 정답이 아닌 이유를 주석문과 코드로 작성
Thing(){}
}

▪나. class Thing {
public Thing(void) {}
}

▪ 다. class Thing {
public Thing(String s) {}
}

▪ 라.public class Thing {
public void Thing() {}
}

```
public class Ex13_03 {
  public static void main(String[] args) {
    new Thing();
  }
}
```


<문제>

4. 다음 중에서 디폴트 생성자로 적합한 것은 어느 것입니까?

```
public class Test { }
```

- (1) Test(void) {}
- (2) public Test(){}
- (3) public Test(void){}
- (4) public void Test() {}