



TIP

이클립스는 생성자를 자동으로 만들어주는 메뉴를 가지고 있다. 클래스 안에 커서를 두고 [Source] → [Generate Constructor using Fields]를 사용해보자. 한 번만 사용해보면 금방 익숙해질 것이다. 생성자에서 초기화할 필드만 체크하면 된다.

TIP





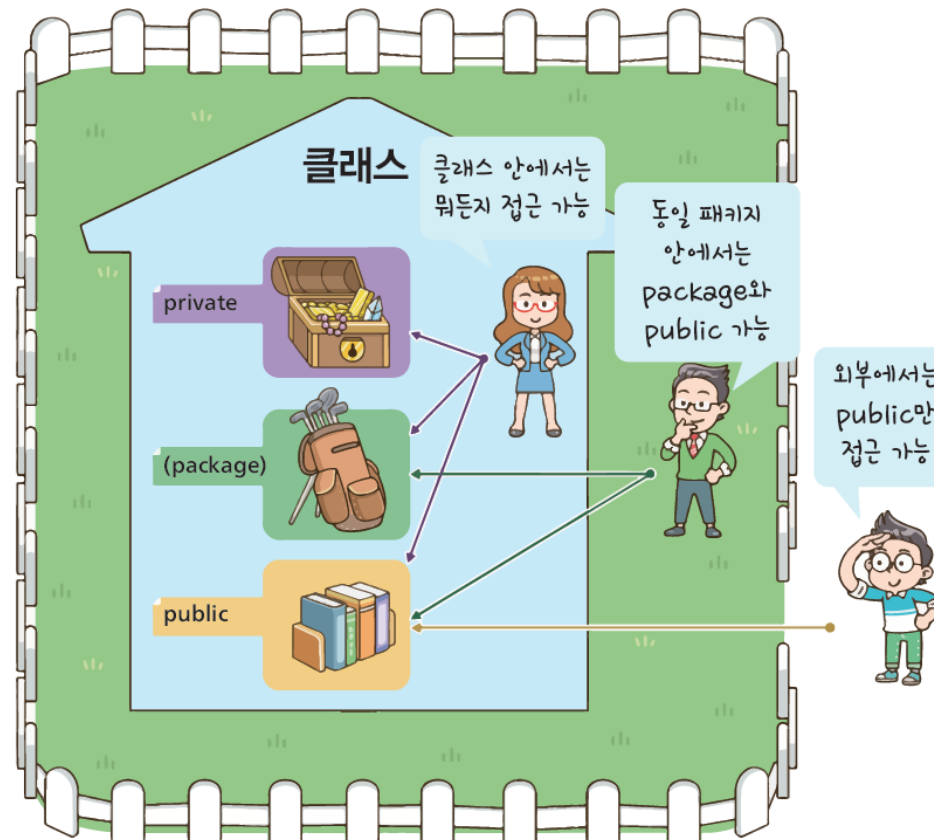
중간점검

1. 만약 클래스 이름이 MyClass라면 생성자의 이름은 무엇이어야 하는가?
2. 생성자의 반환형은 무엇인가?
3. 생성자 안에서 this()의 의미는 무엇인가?
4. this의 주된 용도는 무엇인가?
5. 같은 이름의 메소드를 중복하여 정의하는 것을 _____라고 한다.
6. 메소드 : 오버로딩에서는 무엇으로 이름이 동일한 메소드를 구별하는가?



접근제어

- 접근 제어(access control)란 클래스의 멤버에 접근하는 것을 제어하는 것이다. **public**이나 **private**의 접근 지정자를 멤버 앞에 붙여서 접근을 제한하게 된다





자바의 접근 제어 지정자

접근 지정자	동일한 클래스	패키지	자식 클래스	전체
public	O	O	O	O
protected	O	O	O	X
없음	O	O	X	X
private	O	X	X	X



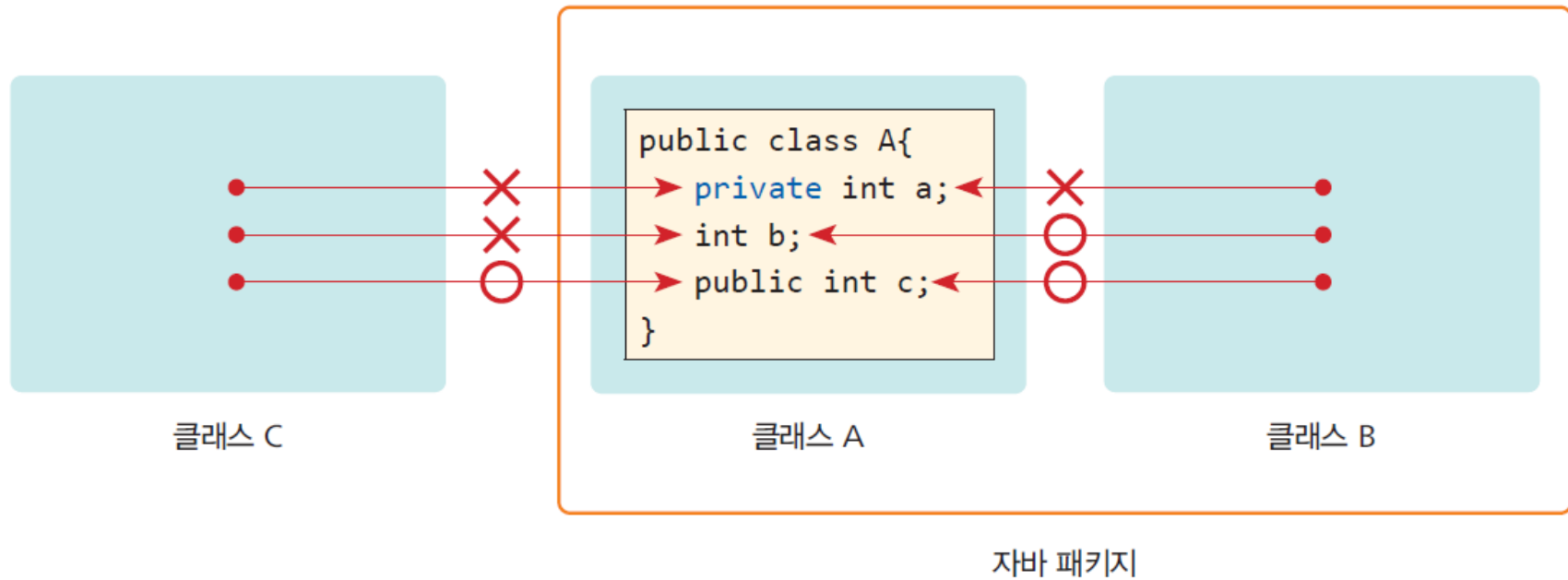
예제

Test.java

```
01  class A {  
02      private int a;        // 전용  
03      int b;                // 디폴트  
04      public int c;         // 공용  
05  }  
06  
07  public class Test {  
08      public static void main(String args[]) {  
09  
10          A obj = new A();    // 객체 생성  
11  
12          obj.a = 10;        // 전용 멤버는 다른 클래스에서는 접근 안 됨  
13          obj.b = 20;         // 디폴트 멤버는 접근할 수 있음  
14          obj.c = 30;         // 공용 멤버는 접근할 수 있음  
15      }  
16  }
```



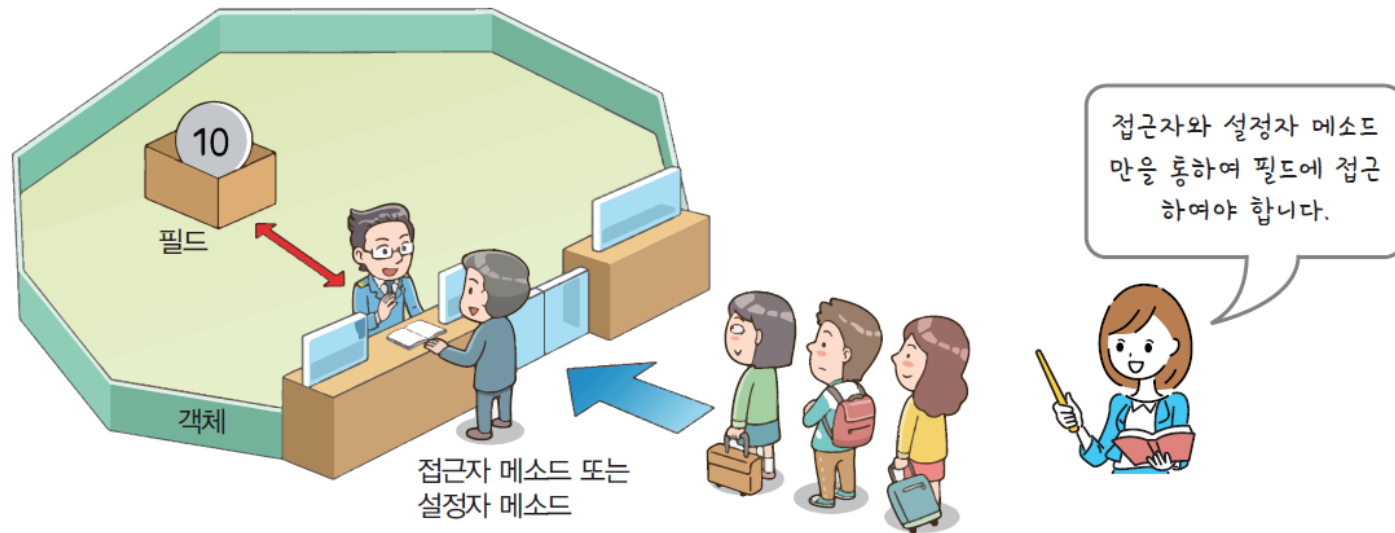
접근 제어





접근자와 설정자

- 필드값을 반환하는 접근자(getters), 필드값을 설정하는 설정자(setters)이다. 이러한 메소드는 대개 **get**이나 **set**이 메소드 이름 앞에 붙여진다. 예를 들면 **getBalance()**는 접근자이고 **setBalance()**는 설정자이다.





접근자와 설정자 예

Account.java

```
01 public class Account {  
02     private int regNumber;  
03     private String name;  
04     private int balance;  
05  
06     public String getName() { return name; }  
07     public void setName(String name) { this.name = name; }  
08     public int getBalance() { return balance; }  
09     public void setBalance(int balance) { this.balance = balance; }  
10  
11 }
```

필드가 모두 private로 선언되었다. 클래스 내부에서만 사용이 가능하다.

접근자와 설정자를 사용하고 있다.

```
12 public class AccountTest {  
13     public static void main(String[] args) {  
14         Account obj = new Account();  
15         obj.setName("Tom");  
16         obj.setBalance(100000);  
17         System.out.println("이름은 " + obj.getName() + " 통장 잔고는 "  
18             + obj.getBalance() + "입니다.");  
19     }  
20 }
```

이름은 Tom 통장 잔고는 100000입니다.



접근자와 설정자를 사용하는 이유

- 접근자와 설정자를 사용해야만 나중에 클래스를 업그레이드할 때 편하다.
- 접근자에서 매개 변수를 통하여 잘못된 값이 넘어오는 경우, 이를 사전에 차단할 수 있다.
- 필요할 때마다 필드값을 동적으로 계산하여 반환할 수 있다.
- 접근자만을 제공하면 자동적으로 읽기만 가능한 필드를 만들 수 있다.

```
public void setAge(int age)
{
    if( age < 0 )
        this.age = 0;
    else
        this.age = age;
}
```



중간점검

1. 필드의 경우, private로 만드는 것이 바람직한 이유는 무엇인가?
2. 필드를 정의할 때 아무런 접근 제어 수식자를 붙이지 않으면 어떻게 되는가?



Lab: 안전한 배열 만들기

- 만약 인덱스가 배열의 크기를 벗어나게 되면 실행 오류가 발생한다. 따라서 실행 오류를 발생하지 않는 안전한 배열을 작성하여 보자.

<<SafeArray>>
-a[]: int +length: int
+get(index: int):int +put(index: int, value: int)



Lab: 안전한 배열 만들기

SafeArrayTest.java

```
01  public class SafeArray {
02      private int a[];
03      public int length;
04
05      public SafeArray(int size) {
06          a = new int[size];
07          length = size;
08      }
09
10      public int get(int index) {
11          if (index >= 0 && index < length) {
12              return a[index];
13          }
14          return -1;
15      }
```



Lab: 안전한 배열 만들기

```
15
16
17 public void put(int index, int value) {
18     if (index >= 0 && index < length) {
19         a[index] = value;
20     } else
21         System.out.println("잘못된 인덱스 " + index);
22 }
23
24 public class SafeArrayTest {
25     public static void main(String args[]) {
26         SafeArray array = new SafeArray(3);
27
28         for (int i = 0; i < (array.length + 1); i++) {
29             array.put(i, i * 10);
30         }
31     }
32 }
```

설정자에서 잘못된 인덱스
번호를 차단할 수 있다.

잘못된 인덱스 3



무엇을 클래스로 만들어야 할까?

- TV, 자동차와 같이, 실제 생활에서 사용되는 객체들은, 클래스로 작성하기가 비교적 쉽다. 하지만 프로그램 안에서는 실제 생활에는 사용되지 않는, 추상적인 객체들도 많이 존재한다.
- 요구 사항이 문서화되면 클래스 식별 과정을 시작할 수 있다. 요구 사항에서 클래스를 식별하는 한 가지 방법은 모든 “명사”를 표시하는 것이다.

은행은 정기 예금 계좌와 보통 예금 계좌를 제공한다. 고객들은 자신의 계좌에 돈을 입금할 수 있으며 계좌에서 돈을 인출할 수 있다. 그리고 각 계좌는 기간에 따라 이자를 지급한다. 계좌마다 이자는 달라진다.





메소드 결정

- 요구 사항 문서에서 “동사”에 해당되는 부분이 메소드가 되는 경우가 많다. 앞의 요구 사항 문서에서 “고객들은 자신의 계좌에 돈을 입금할 수 있으며 계좌에서 돈을 인출할 수 있다.”라는 문장이 있다. 여기서 “입금한다”, “인출한다”와 같은 동사는 **deposit()**, **withdraw()** 메소드로 매핑시킬 수 있다.

은행은 정기 예금 계좌와 보통 예금 계좌를 제공한다. 고객들은 자신의 계좌에 돈을 입금할 수 있으며 계좌에서 돈을 인출할 수 있다. 그리고 각 계좌는 기간에 따라 이자를 지급한다. 계좌마다 이자는 달라진다.





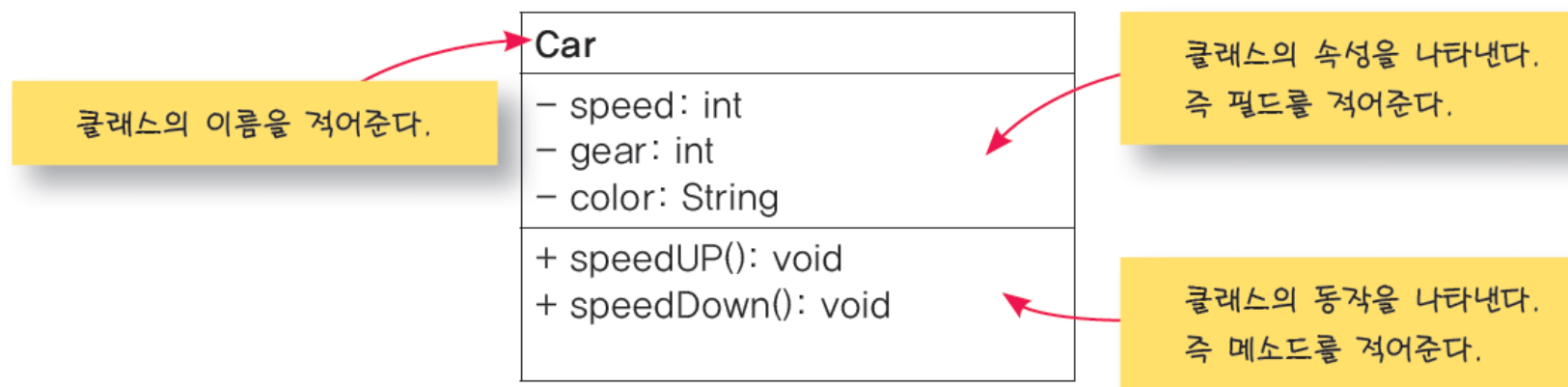
클래스 간의 관계를 결정한다.

- 이러한 클래스 간의 관계를 설정하는 부분은 객체 지향 설계에서 가장 어려운 부분이다. 개발자는 많은 결정을 내려야 한다. 가장 중요한 결정은 다음의 2가지이다.
 - 상속(Inheritance)
 - 구성(Composition)



UML

- UML(Unified Modeling Language): UML은 클래스만을 그리는 도구는 아니고 객체지향설계 시에 사용되는 일반적인 모델링 언어라고 할 수 있다.
- UML을 사용하면 소프트웨어를 본격적으로 작성하기 전에 구현하고자 하는 시스템을 시각화하여 검토할 수 있다.





가시성 표시자








- 필드나 메소드의 이름 앞에는 가시성 표시자(visibility indicator)가 올 수 있다.
- +는 public을, -는 private을 의미한다.

+	Public
-	Private
#	Protected
/	Derived
~	Package



클래스 간의 관계

표 4-1 UML에서 사용되는 화살표의 종류

관계	화살표
일반화(generalization), 상속(inheritance)	
구현(realization)	
구성관계(composition)	
집합관계(aggregation)	
유향 연관(direct association)	
양방향 연관(bidirectional association)	
의존(dependency)	



의존 관계

- 의존(dependency)이란 하나의 클래스가 다른 클래스를 사용하는 관계이다.

CarTest.java

```
01 public class CarTest {  
02     public static void main(String[] args) {  
03         Car myCar = new Car();  
04         myCar.speedUp();  
05     }  
06 }
```

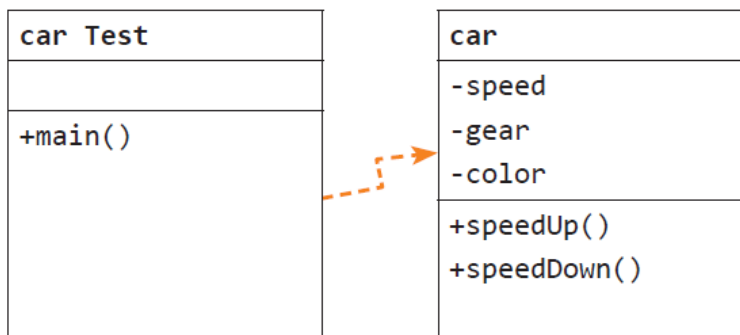


그림 4.6 Car 예제의 UML



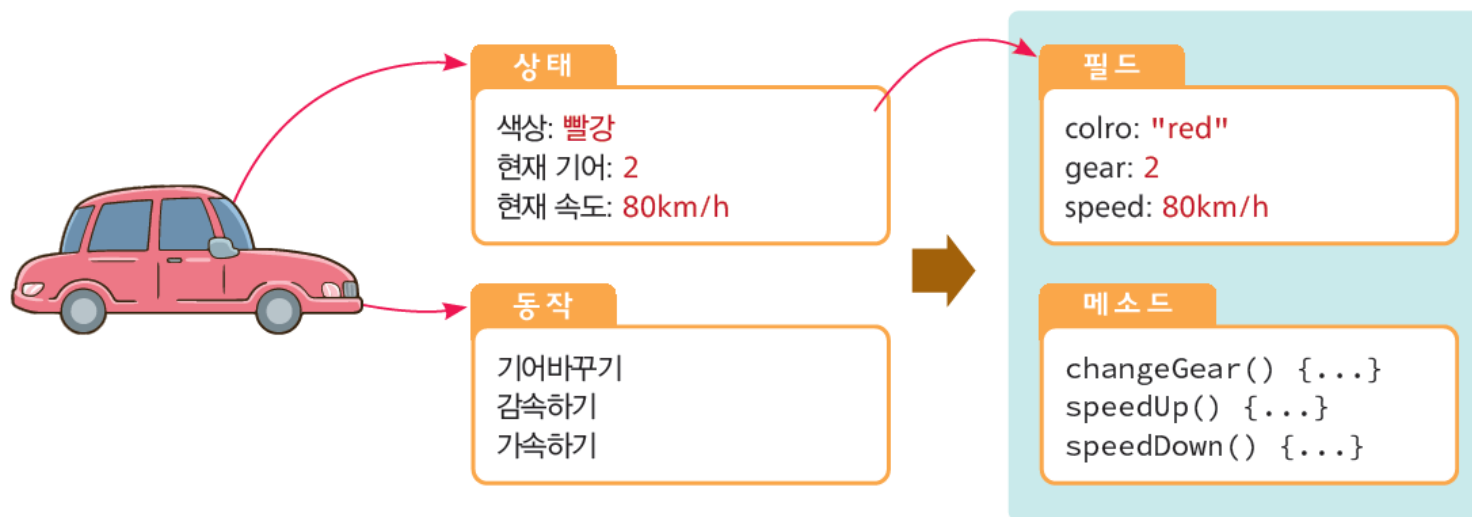
중간점검

1. 작업 명세서가 주어졌다고 하자. 클래스 후보는 어떻게 찾을 수 있는가?
2. 작업 명세서가 주어졌다고 하자. 메소드 후보는 어떻게 찾을 수 있는가?
3. TV를 나타내는 클래스를 정의하고 UML의 클래스 다이어그램으로 표현하여 보라.



Lab: 자동차 클래스 작성

- 자동차를 나타내는 클래스를 정의하여 보자. 예를 들어, 자동차 객체의 경우 속성은 색상, 현재 속도, 현재 기어 등이다. 자동차의 동작은 기어 변속하기, 가속하기, 감속하기 등이 있다.





Sol:

CarTest.java

```
01 class Car {
02     String color; // 색상
03     int speed;    // 속도
04     int gear;     // 기어
05
06     @Override
07     public String toString() {
08         return "Car [color=" + color + ", speed=" + speed + ", gear=" + gear + "]";
09     }
10
11     void changeGear(int g) {         gear = g;         }
12     void speedUp() {                 speed = speed + 10; }
13     void speedDown() {               speed = speed - 10; }
14 }
15
16 public class CarTest {
17     public static void main(String[] args) {
18
19         Car myCar = new Car();
20         myCar.changeGear(1);
21         myCar.speedUp();
22         System.out.println(myCar);
23     }
24 }
```

toString() 메소드는 이클립스에서 자동으로 생성시킬 수 있다. [Source] → [Generate toString()...] 메뉴를 사용해보자.

Car [color=null, speed=10, gear=1]



Lab: 은행 계좌 클래스 작성

- 은행 계좌를 클래스로 정의하여 보자. 먼저 잔액은 변수 **balance**로 표시한다. 예금 입금은 **deposit()** 메소드로, 예금 인출은 **withdraw()** 메소드로 나타내면 된다.

BankAccount
-owner : string -accountNumber : int -balance : int
+deposit() +withdraw()





Sol:

BankAccount.java

```
01 class BankAccount { // 은행 계좌
02     int accountNumber; // 계좌 번호
03     String owner; // 예금주
04     int balance; // 잔액을 표시하는 변수
05
06     void deposit(int amount) {          balance += amount;   }
07     void withdraw(int amount) {         balance -= amount;   }
08     public String toString(){
09         return "현재 잔액은 " + balance + "입니다.";
10     }
11 }
12
13 public class BankAccountTest {
14     public static void main(String[] args) {
15         BankAccount myAccount = new BankAccount();
16         myAccount.deposit(10000);
17         System.out.println(myAccount);
18         myAccount.withdraw(8000);
19         System.out.println(myAccount);
20
21     }
22 }
```

현재 잔액은 10000입니다.

현재 잔액은 2000입니다.



Lab: 윈도우 생성해보기

- 자바에서 윈도우를 나타내는 클래스는 **JFrame**이다. 윈도우를 만들 때 가장 힘든 작업이 윈도우의 기본 값들을 설정하는 부분이다. 객체 지향 방법에서는 “생성자” 개념이 있어서 이 부분이 아주 쉽게 가능하다. 개발자가 아무 것도 설정하지 않아도 생성자에서 기본값으로 설정된다





Sol: 윈도우 생성해보기

FrameTest.java

01 **import** javax.swing.*;

① javax.swing이라는 패키지
안의 모든 클래스를 읽어들인다.

03 **public class** FrameTest {

04 **public static void** main(String[] args) {

05 JFrame f = **new** JFrame("Frame Test");

② JFrame 클래스의 객체를 생
성하고, 생성자를 호출한다.

06

07 f.setSize(300, 200);

08 f.setVisible(**true**);

③ JFrame 클래스의 메소드를
호출한다.

09

}

10

}



Mini Project: 주사위 클래스

- 주사위를 **Dice** 클래스로 모델링한다. **Dice** 클래스는 주사위면(**face**)을 필드로 가지고 있고 **roll()**, **getValue()**, **setValue()** 등의 메소드를 가지고 있다. 생성자에서는 주사위면을 0으로 초기화한다.



주사위1= 5 주사위2= 5
주사위1= 3 주사위2= 4
주사위1= 1 주사위2= 1
(1, 1)이 나오는데 걸린 횟수= 3



Summary

- 객체 지향 방법은 실세계가 객체로 구성되어 있는 것처럼 소프트웨어도 객체로 구성하는 방법론이다.
- 절차 지향은 함수를 사용하여 프로그램을 작성하는 방법이다. 함수는 재사용하기가 어렵다.
- 객체 지향을 이루고 있는 핵심적인 개념에는 “캡슐화”, “상속”, “다형성”, “추상화” 등이 있다.
- 캡슐화는 객체의 속성과 동작을 하나로 묶는 것을 의미한다. 내부 구현을 알 수 없게 은닉하는 것도 캡슐화에 포함된다.
- 상속은 다른 클래스를 재사용하는 강력한 방법이다.
- 객체는 속성과 동작으로 정의된다. 프로그램에서는 속성은 필드로, 동작은 메소드로 구현된다.
- 자바에서는 **new**를 사용하여 객체를 생성한다.
- 객체를 생성하기 전에 반드시 객체를 참조하는 변수를 먼저 선언하여야 한다.
- 메소드 오버로딩이란 이름은 동일하지만 매개 변수가 다른 메소드를 여러 개 정의하는 것이다.





Q & A

