

Classes and Objects: A Trip to Objectville

Samkeun Kim <skim@hknu.ac.kr>

<http://cyber.hankyong.ac.kr>



We're going to
Objectville! We're
leaving this dusty ol'
procedural town for good.
I'll send you a postcard.

Chair Wars:

(or How Objects Can Change Your Life)



누가 의자를 차지할 것인가?

옛날 옛적에 소프트웨어 샵에서 두 명의 프로그래머에게 동일한 스펙을 부여하고 "빌드"하라는 명령을 내렸다.

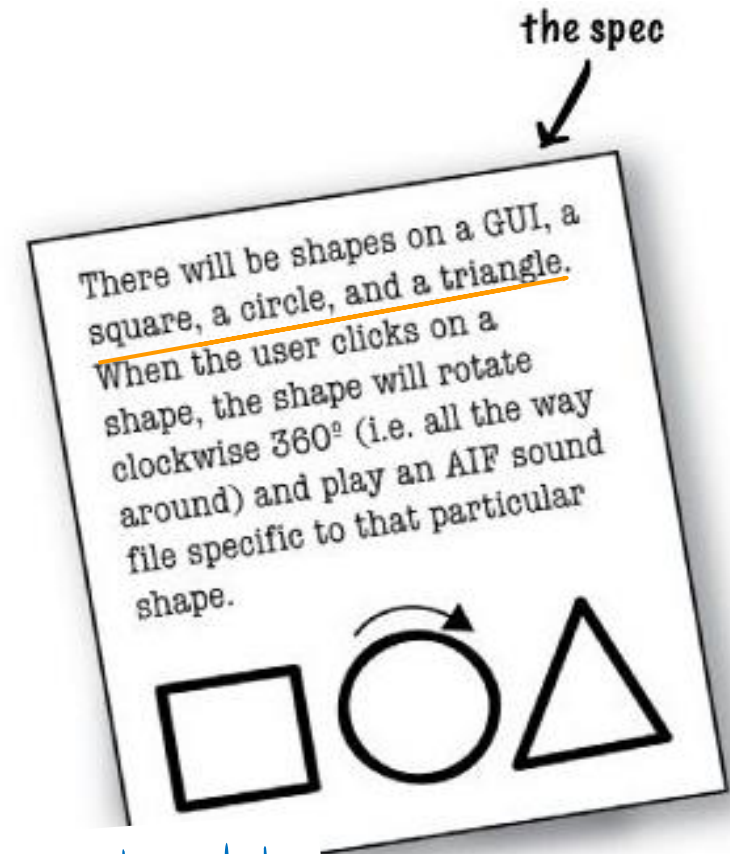
Really Annoying Project Manager는 두 Coder에게 경쟁을 시켰다:

먼저 구축하는 사람에게 실리콘 밸리의 모든 사람들이 갖고 싶어하는 Cool한 Aeron™ 의자 하나를 제공하겠다고 약속했다.

Larry => Procedural Programmer

Brad => OO Guy

GUI에 사각형, 원, 삼각형 모양이 있다.
사용자가 한 모양을 클릭하면 그 모양을
시계방향으로 360° 회전(rotate)시키고
해당 모양에 연결된 AIF 사운드 파일을
play하시오(playSound).



Larry/Brad의 생각 차이

Larry, 조그마한 사무실에 앉아서 곰곰이 생각해 보았다.

“이 프로그램이 수행해야 할 것들이 무엇인가? 어떤 프로시저들이 필요할까?”

그는 스스로 답했다: “**rotate and playSound.**”

⇒ 그래서 곧바로 프로시저들을 구현했다.

⇒ 프로그램이 프로시저들을 모아 놓은 것이 아니라면 무엇이란 말인가?

Brad, 카페에 앉아 긴장을 풀면서 곰곰이 생각해 보았다:

"What are the things in this program... who are the key players?"

그는 모양들^{Shapes}을 먼저 생각했다.

물론 User, Sound, Clicking Event와 같은 다른 객체들도 있다.

그러나 이런 것들은 라이브러리 코드에 이미 들어 있다.

⇒ *Shapes를 구축하는 데에 집중하자!!*

누가 의자를 차지했나?

Brad와 **Larry**가 프로그램을 어떻게 구축했는지 읽어보고,

"그래서 누가 Aeron을 얻을까?"

라는 질문에 대한 답변을 추측해보자.

"So, who got the Aeron?"



↖
the chair

In Larry's cube

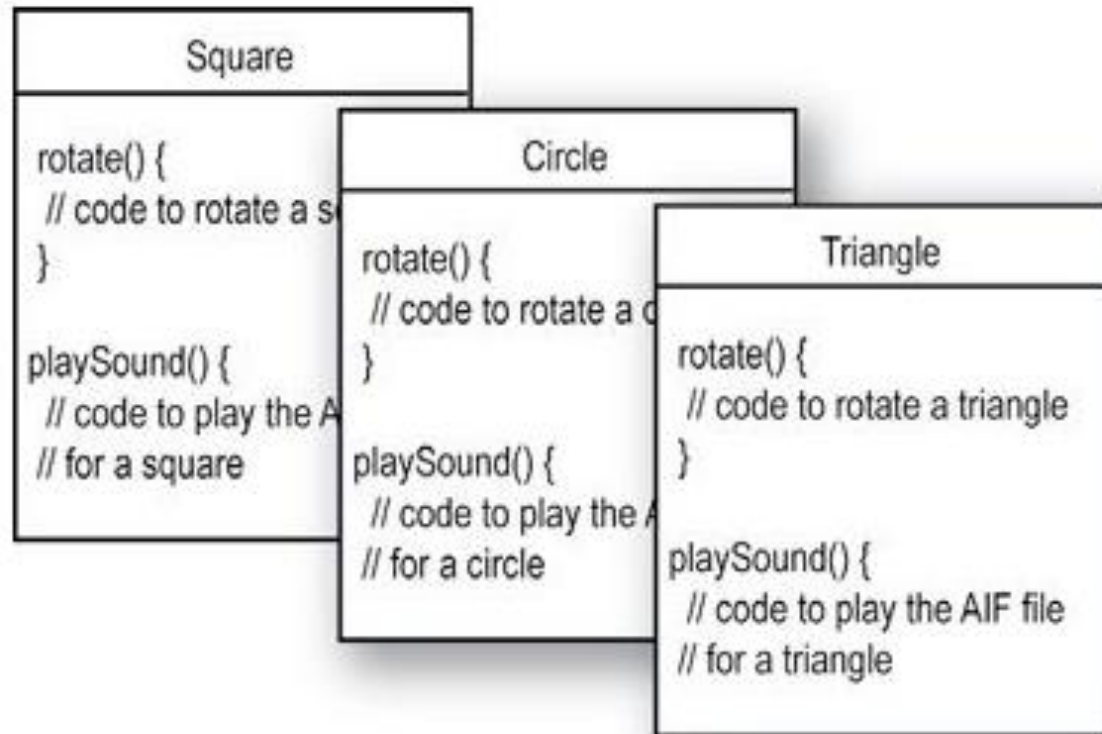
Larry는 예전부터 늘 해오던 방식대로 거침없이 주요 프로시저를 작성했다:

- rotate / playSound

```
rotate(shapeNum) {  
    // make the shape rotate 360°  
}  
playSound(shapeNum) {  
    // use shapeNum to lookup which  
    // AIF sound to play, and play it  
}
```


At Brad's laptop at the cafe

Brad는 세 가지 모양 각각에 대해 클래스를 작성했다.



Larry thought he'd nailed it. He could almost feel the rolled steel of the Aeron beneath his...

잠깐! 기다려라. 그 사이 스펙 변경이 있었다.

오케이. 기술적으로는 **Larry**가 먼저 이겼다. 그러나 프로그램에 한 가지 사소한 것을 추가해야만 한다.

PM은 말한다: "당신 둘처럼 실력있는 프로그래머에게는 전혀 문제될게 없을 것이다" 라고.

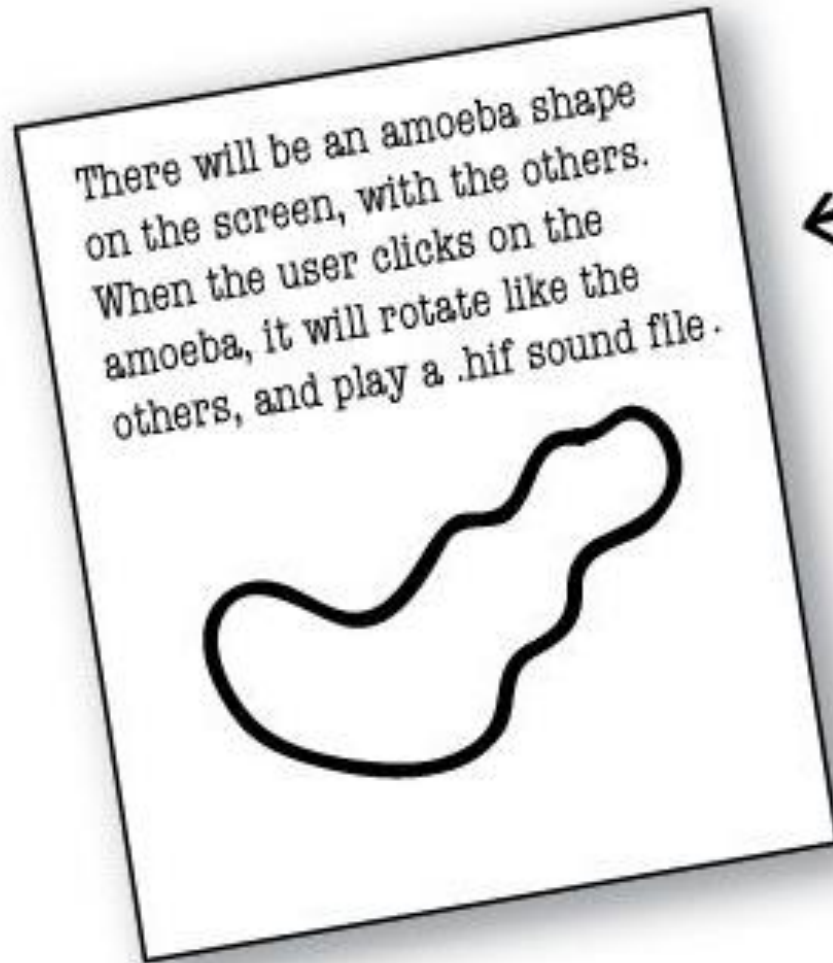
Larry는 스펙이 변경된 사실에 매우 불만이다: "*spec-change-no-problem was a fantasy*"

Brad는 이상하게 평화롭다. 스펙이 변경된 것이 뭐가 문제가 되는데?

“스펙은 항상 바뀐다”

— 모든 프로그래머 —

But wait! There's been a spec change



← what got added to the spec

GUI에 아메바 모양이 추가된다.

기존의 다른 모양들과 마찬가지로 사용자가 아메바 모양을 클릭하면 시계방향으로 360° 회전(rotate)시키고 아메바 모양에 연결된 .hif 사운드 파일을 Play하시오(playSound).

Larry의 사무실로 돌아와 보자!

Rotate() 프로시저는 여전히 동작한다:

룩업 테이블을 이용하여 **shapeNum**에 매치하는 모양^{Shapes}을 찾을 수 있다!

그러나 **playSound()**는 변경해야만 한다 => **.hif** 파일 연결?

```
playSound(shapeNum) {  
    // if the shape is not an amoeba,  
    // use shapeNum to lookup which  
    // AIF sound to play, and play it  
    // else  
    // play amoeba .hif sound  
}
```

해변가의 Brad's laptop에서는...

Brad: 단순히 새로운 클래스를 하나 더 작성했다 (*Amoeba* 클래스)

Brad가 객체지향 방법론(OO)에 관해 가장 사랑하는 것 중의 하나가 바로 이미 테스트된 코드를 전혀 건드리지 않아도 된다는 사실이다.

"Flexibility, extensibility,..." he mused, reflecting on the benefits of OO.

Amoeba
<pre>rotate() { // code to rotate an amoeba } playSound() { // code to play the new // <u>.hif file for an amoeba</u> }</pre>

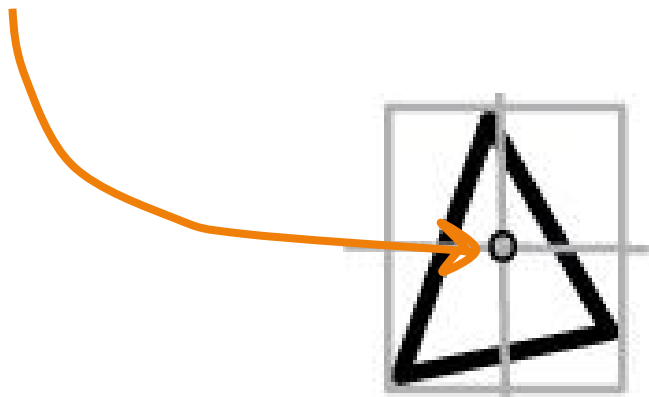
그래도 Larry가 Brad를 앞질렀다.

정말로 짜증나게 하는 프로젝트 매니저가 실망한 투로 "Oh, no, that's not how the amoeba is supposed to rotate..." 라고 말하기 전 까지만!!

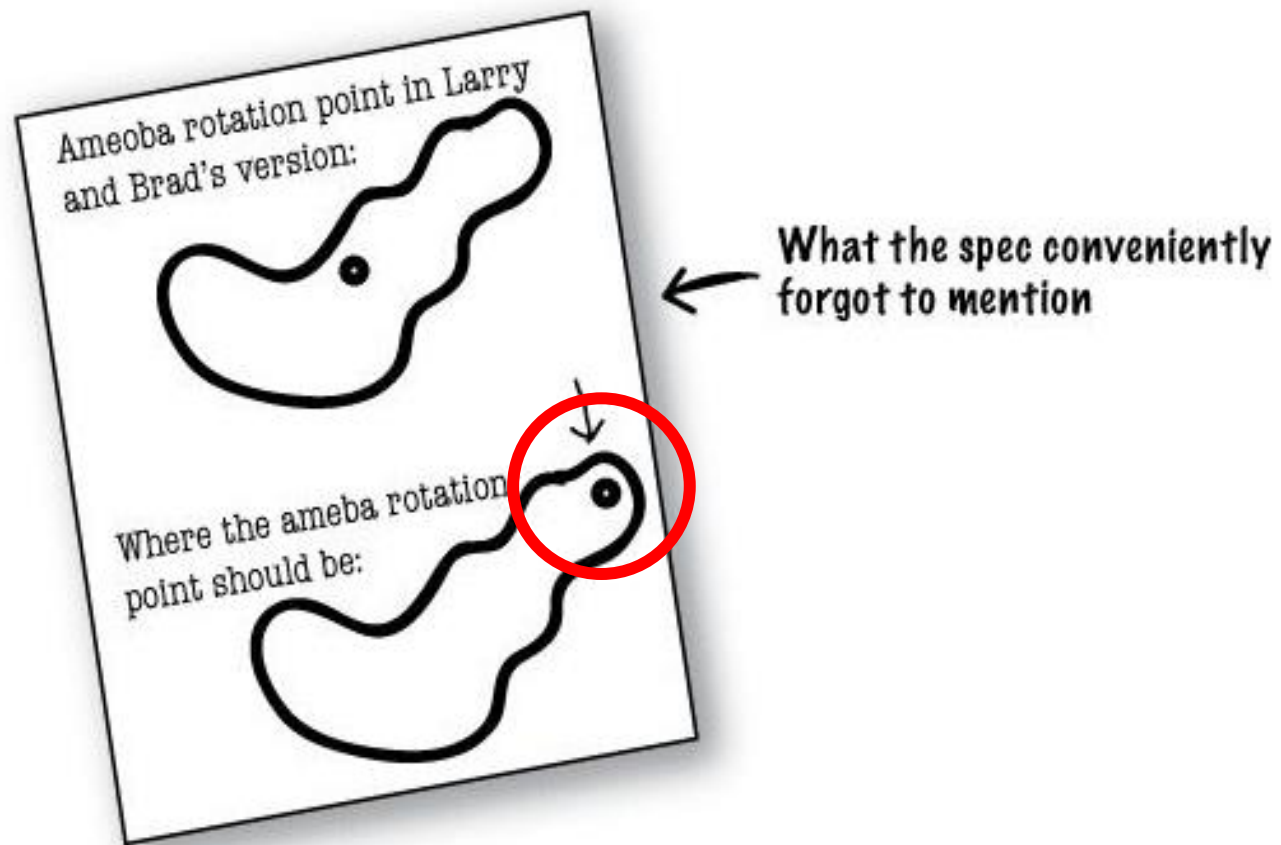
Larry의 얼굴은 굳어졌다.

결국 두 프로그래머 모두 아래와 같이 메소드를 작성했던 것으로 밝혀졌다:

1. 모양을 감싸는 사각형을 그린다.
2. 사각형의 중심을 계산하여 그 점을 중심으로 모양을 회전시킨다.



그러나 *amoeba* 모양은 시계 추처럼 한쪽 끝에서 회전시켜야 한다고 요구하고 있다.



Larry의 사무실로 돌아와서. . .

Larry: 그는 회전점(rotate point)을 **rotate()** 프로시저의 아규먼트로 전달하면 될 거라고 생각했다.

⇒ 많은 코드 부분이 영향을 받았다: Testing, recompiling, ...

(9야드 정도 길이를 수정해야...)

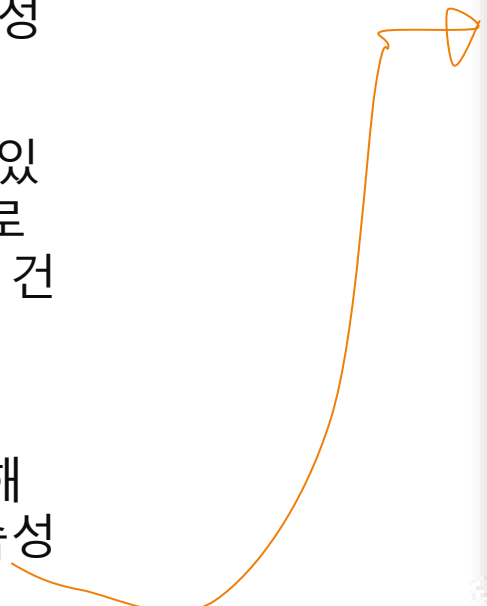
⇒ 예전에 잘 작동되던 것들이 이젠 동작하지 않는다 . . .

At Brad's laptop on his lawn chair

Brad:

브래드 역시 **rotate()** 메소드를 수정해야 하지만 **Amoeba** 클래스에서만 수정하면 된다:

- ⇒ 이미 테스트되었고 작동되고 있고 컴파일 되어 있는 기존 프로그램의 코드에 대해서는 전혀 건드릴 필요가 없다
- ⇒ 다만 Amoeba에게 회전점 (rotate point)을 제공하기 위해 모든 아메바가 가져야 하는 속성을 추가했다



Amoeba
int xPoint
int yPoint
rotate() { // code to rotate an amoeba // using amoeba's x and y }
playSound() { // code to play the new // .hif file for an amoeba }

그래서 Brad가 의자를 차지했겠죠?

너무 성급한 결론이다!

Larry는 **Brad**의 접근 방식에 결함을 발견했다. 그리고 의자를 획득하면 회계부의 루시의 관심도 얻을 걸로 확신했다. 분위기를 반전시켜야만 한다!

LARRY: 중복 코드가 있다! **Rotate** 프로시저가 4가지 **Shape**라는 것(things) 모두에 중복되어 있다.

BRAD: 프로시저가 아니라 **메소드**이다. 그리고 사물(thing)이 아니라 **클래스**이다.

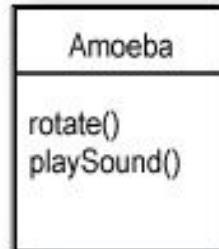
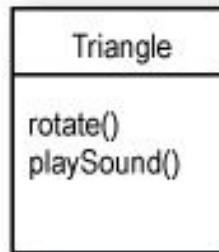
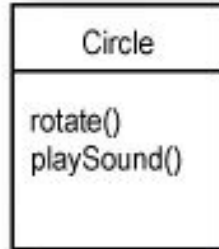
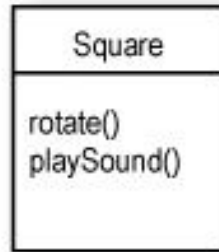
LARRY: 뭐든 간에. 이것은 바보 같은 디자인이다. 4가지 회전 "methods"를 유지해야 한다. 그게 어떻게 좋다고 말할 수 있는가?

BRAD: 아, 최종 디자인을 못 봐서 그런 것 같다. 내가 OO 상속이 어떻게 작동하는지 보여주마!



What Larry wanted ↗
(figured the chair would impress her)

Brad's Final Design



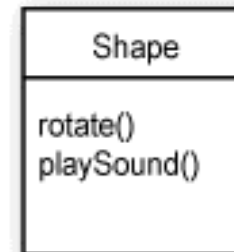
1

I looked at what all four classes have in common.



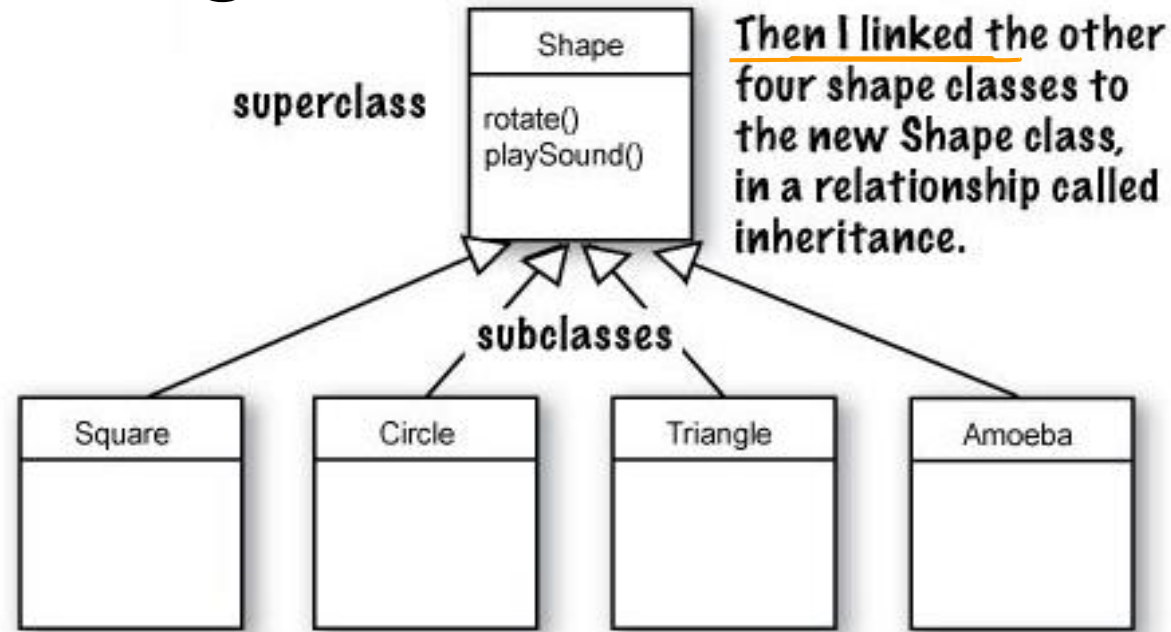
2

They're Shapes, and they all rotate and playSound. So I abstracted out the common features and put them into a new class called Shape.



Brad의 Final Design

3



서브 클래스는 슈퍼 클래스
의 기능 (functionality)
을 자동으로 물려받게 된다!

다음처럼 읽을 수 있다:

"Square inherits from Shape", "Circle inherits from Shape" 등

여러 다른 모양으로부터 **rotate()**와 **playSound()**를 제거했기 때문에 이제 오로지 하나의 복사본만 관리하면 된다

Shape 클래스는 다른 4 개 클래스의 슈퍼 클래스이다.

다른 4 개의 클래스는 **Shape**의 서브 클래스가 된다.

그렇다면 Amoeba 회전은 어떻게 시키나?

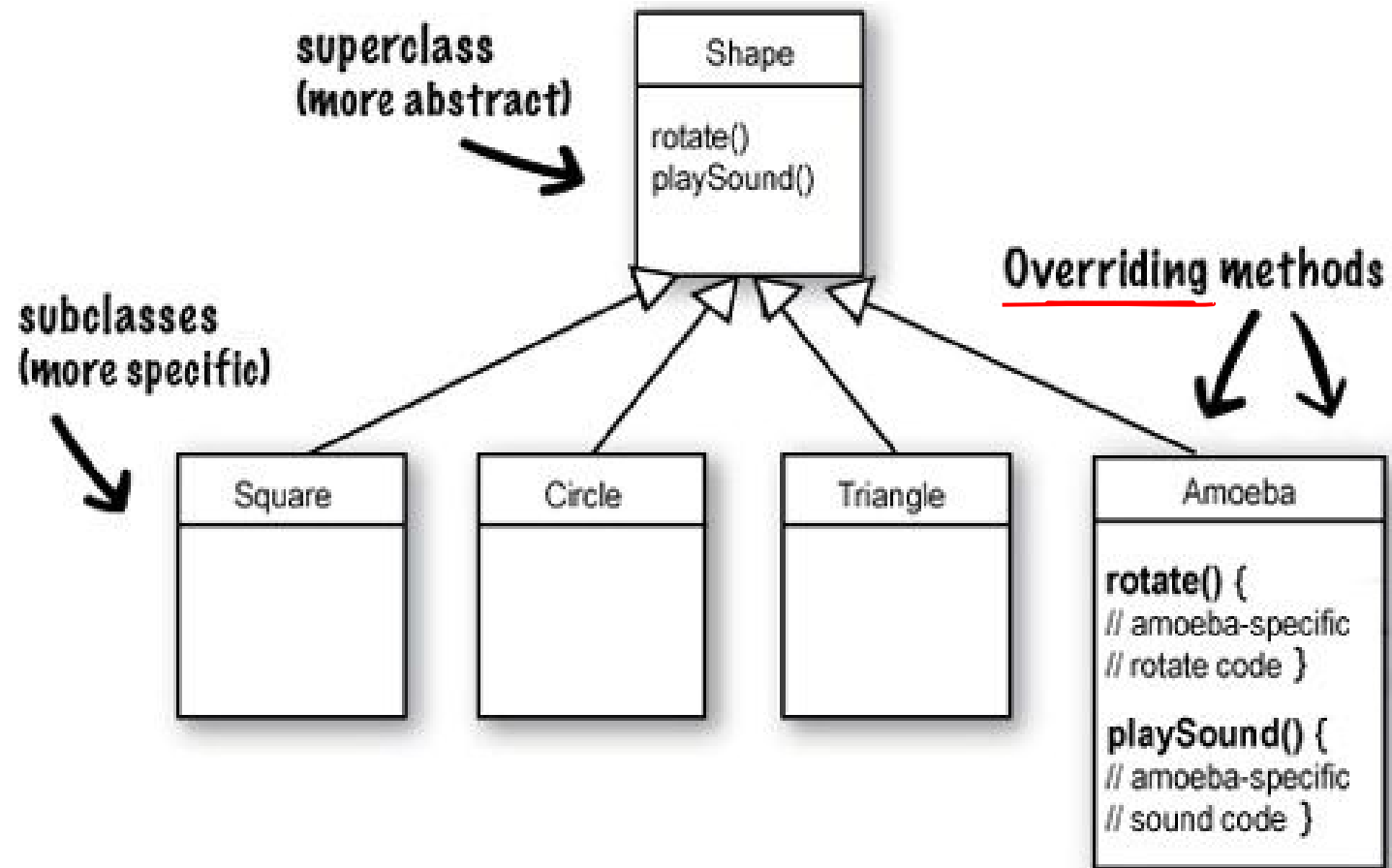
LARRY: 문제는 그게 다가 아니다. 아메바 모양은 완전히 다른 **rotate**와 **playSound** 프로시저를 가지고 있지 않은가?

BRAD: 메소드!!

LARRY: 뭐든 간에. **Shape** 클래스로부터 기능을 상속받는다면 아메바만의 다른 점은 어떻게 처리하나?



BRAD: 그것이 마지막 단계다. **Amoeba** 클래스는 **Shape** 클래스의 메소드를 오버라이딩한다 (override). JVM은 정확히 어느 **rotate()** 메소드를 실행시킬 지를 알고 있다.



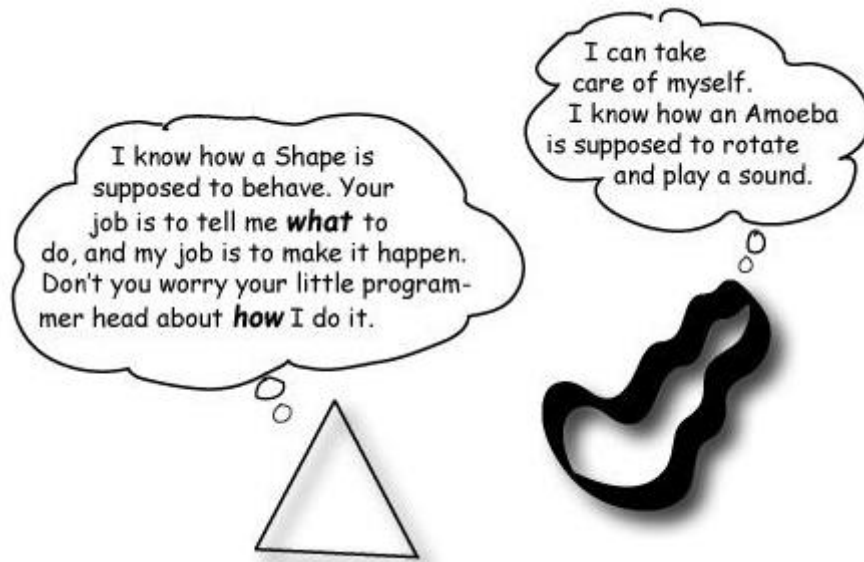
4

I made the Amoeba class override the `rotate()` and `playSound()` methods of the superclass Shape.

Overriding just means that a subclass redefines one of its inherited methods when it needs to change or extend the behavior of that method.

LARRY:

아메바에게 뭔가를 수행하도록 어떻게 "말해주느냐"?
프로시저, 아니 메소드를 호출해야 하지 않나?
그런데 어느 클래스에 있는 **rotate()**를 호출하도록 해야 하지?



BRAD:

이것이 바로 **OO**의 Cool한 면이다.
예를 들어, 삼각형을 회전시켜야 할 때가 오면 프로그램 코드는 삼각형 객체의 **rotate()** 메소드를 호출한다.
프로그램의 나머지는 삼각형 객체가 회전을 어떻게 시키는지 알지도 못하고 관심도 없다.
즉, 프로그램에 뭔가 새로운 것을 추가시킬 필요가 있을 때는 단순히 그 새로운 객체 타입에 대한 새로운 클래스를 작성하면 된다.
⇒ 새로운 객체: 자신만의 **Behavior**를 갖는다!

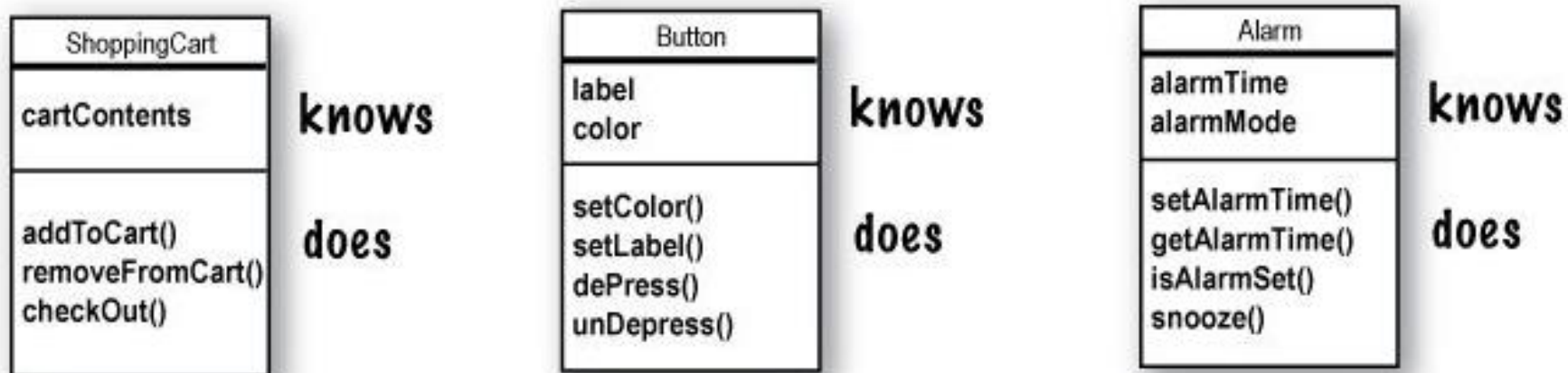
The suspense is killing me. Who got the chair?



클래스 설계: 클래스 타입으로부터 생성될 객체를 생각하라

Think about

- Things the object **knows**
- Things the object **does**



Snooze button:

스누즈 버튼(아침에 잠이 깬 뒤 조금 더 자기 위해 누르는 라디오나 시계의 타이머 버튼)

객체가 자신에 관해 알고 있는 것:

- **인스턴스 변수** (instance variables)

객체가 수행할 수 있는 것:

- **메소드** (methods)

**instance
variables**
(state)

methods
(behavior)



knows

does

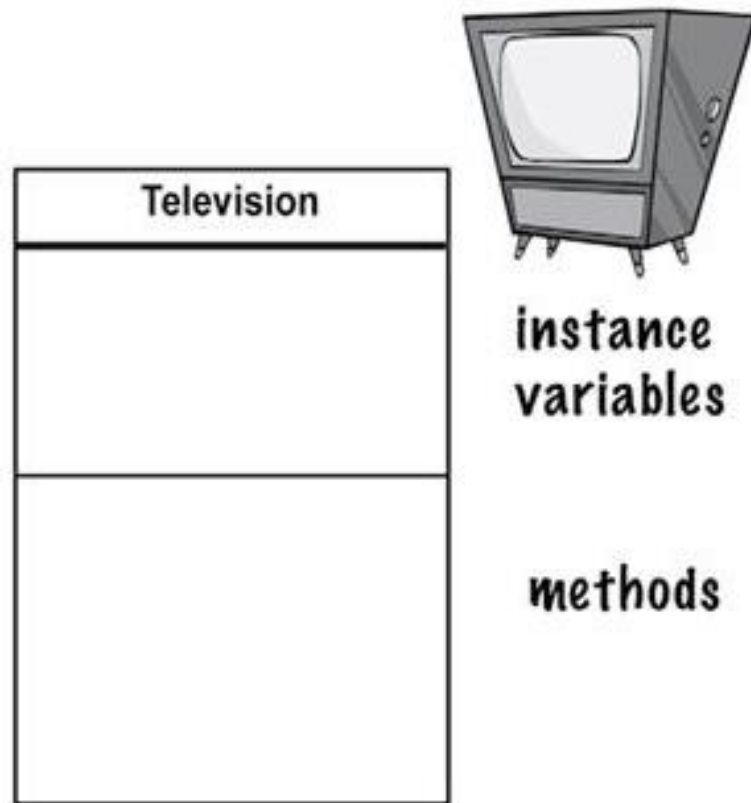
Things an object knows about itself are called instance variables. They represent an object's state (the data), and can have unique values for each object of that type.

인스턴스 => 객체를 의미하는 또 다른 용어라고 생각하면 된다.

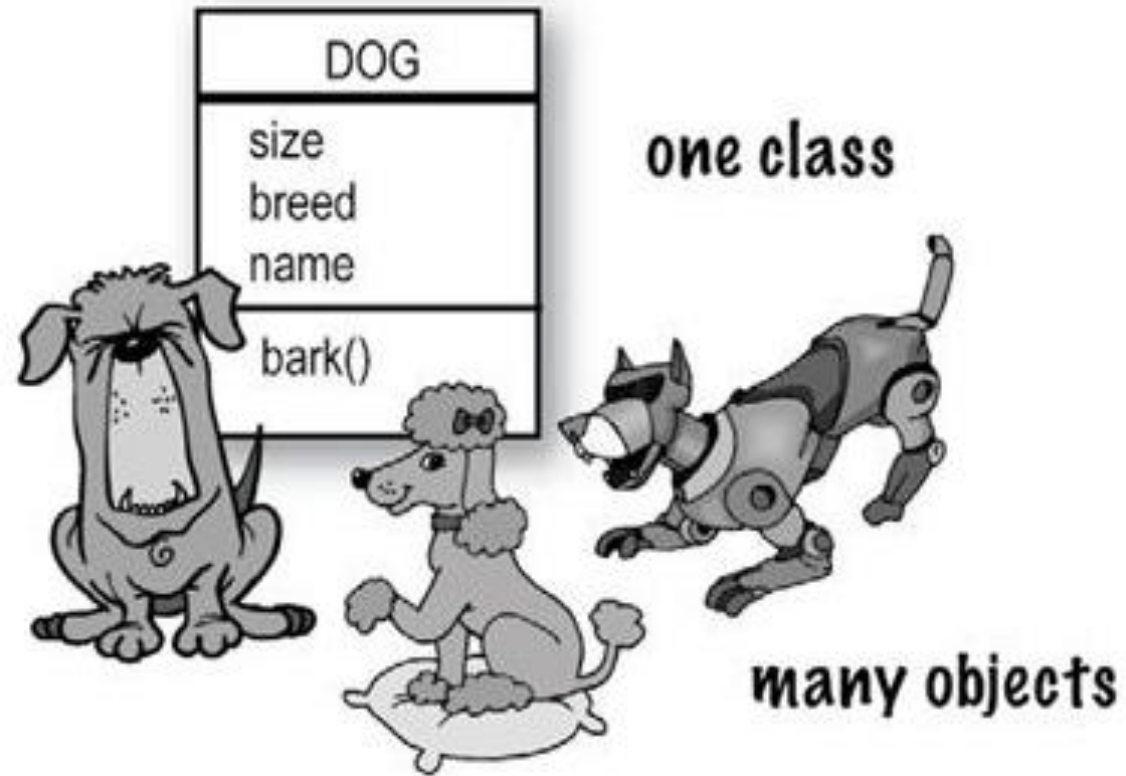
- 객체가 할 수 있는 것을 **메소드**라고 부른다.
- 클래스를 설계할 때
 - 객체가 알 필요가 있는 데이터를 고려해야
 - 또한 그 데이터 상에서 운용될 메소드를 설계해야
- 객체가 인스턴스 변수의 값을 **read** 하고 **write** 하는 메소드를 가져야 하는 것은 공통된 부분
- 예를 들어, **Alarm** 객체는 **alarmTime**을 저장할 인스턴스 변수와 **alarmTime**을 얻어 오고(getting), 세팅(setting)할 2개의 메소드를 가져야
- 따라서 객체는 인스턴스 변수와 메소드를 가지며 둘 다 클래스의 일부분으로 고려해야

실습과제 2-1 Sharpen your pencil

Fill in what a television object might need to know and do.



What's the difference between a class and an object?

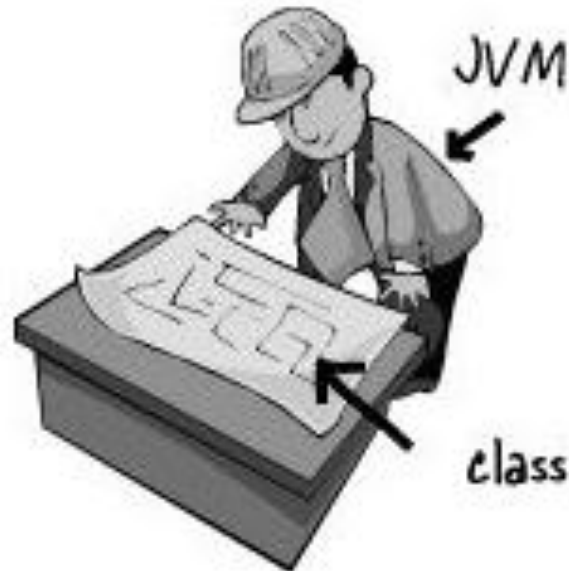


A class is not an object.

(but it's used to construct them)

클래스 => 객체에 대한 청사진 (blueprint)

- 특정 타입의 객체를 어떻게 만들 것인가를 JVM에게 알려준다.
- 클래스로부터 만들어진 각 객체는 클래스의 인스턴스 변수에 대한 자신만의 값을 가진다.
- 예를 들어, **Button** 클래스를 이용하여 수십 개의 서로 다른 버튼들을 만들어 낼 수 있고, 각 버튼 객체는 자신만의 색깔, 사이즈, 모양 등을 가질 수 있다.



LOOK AT IT THIS WAY...



객체는 주소록에 있는 하나의 항목과 같다.

One analogy for objects is a packet of unused Rolodex™ cards. Each card has the same blank fields (the instance variables). When you fill out a card you are creating an instance (object), and the entries you make on that card represent its state. The methods of the class are the things you do to a particular card; getName(), changeName(), setName() could all be methods for class Rolodex. So, each card can do the same things (getName(), changeName(), etc.), but each card knows things unique to that particular card.

Making your first object

Dot(.) operator

THE DOT OPERATOR (.)

The dot operator (.) gives you access to an object's state and behavior (instance variables and methods).

// make a new object

Dog d = new Dog();

// tell it to bark by using the

// dot operator on the

// variable *d* to call bark()

d.bark();

// set its size using the

// dot operator

d.size = 40;

객체를 생성하고 사용하기 위해 무엇이 필요한가

일반적으로 2개의 클래스가 필요하다:

- 사용하고 싶은 객체 타입에 대한 클래스
(Dog, AlarmClock, Television, etc.)
- 새로운 클래스를 테스트하기 위한 클래스
 - ✓ 테스터 클래스 => **main()** 메소드를 가져야 한다.
main() 메소드에서 새로운 클래스 타입의 객체를 생성(create)하고 접근(access)한다.
 - ✓ 테스터 클래스 => 새로운 객체의 메소드와 변수를 수행하는 일만 한다.

Dog class

1. Write your class (*Dog.java*)

```
class Dog {  
  
    int size;  
    String breed;  
    String name;  
  
    void bark() {  
        System.out.println("Ruff! Ruff!");  
    }  
}
```

instance variables

a method

DOG
size breed name
bark()

2. Write a tester (*DogTestDrive*) class

just a main method
(we're gonna put code
in it in the next step)

```
class DogTestDrive {  
    public static void main (String[] args) {  
        // Dog test code goes here  
    }  
}
```

3. In your tester, make an object and access the object's variables and methods (*DogTestDrive.java*)

```
class DogTestDrive {  
    public static void main (String[] args) {  
        Dog d = new Dog();  
        d.size = 40;  
        d.bark();  
    }  
}
```

make a Dog object

use the dot operator (.)
to set the size of the Dog
and to call its bark() method

dot
operator

Making and testing Movie objects



```
class Movie {
    String title;
    String genre;
    int rating;

    void playIt() {
        System.out.println("Playing the movie");
    }
}

public class MovieTestDrive {
    public static void main(String[] args) {
        Movie one = new Movie();
        one.title = "Gone with the Stock";
        one.genre = "Tragic";
        one.rating = -2;
        Movie two = new Movie();
        two.title = "Lost in Cubicle Space";
        two.genre = "Comedy";
        two.rating = 5;
        two.playIt();
        Movie three = new Movie();
        three.title = "Byte Club";
        three.genre = "Tragic but ultimately uplifting";
        three.rating = 127;
    }
}
```

한 파일에 여러 클래스를 선언
할 경우 **public** 클래스는 하나
만 있어야 된다!

Quick! Get out of main!

main() 의 2가지 용도

main()에만 빠져 있는 한 진정으로 객체마을에 있는 것이 아니다!

테스트 프로그램이 **main()** 메소드 내에서 실행되는 것은 좋지만 진정한 객체지향 애플리케이션에서는 서로 다른 객체들끼리 상호작용한다.

이것은 **static main()** 메소드가 일방적으로 객체를 생성하고 테스트하는 것과는 반대된다.

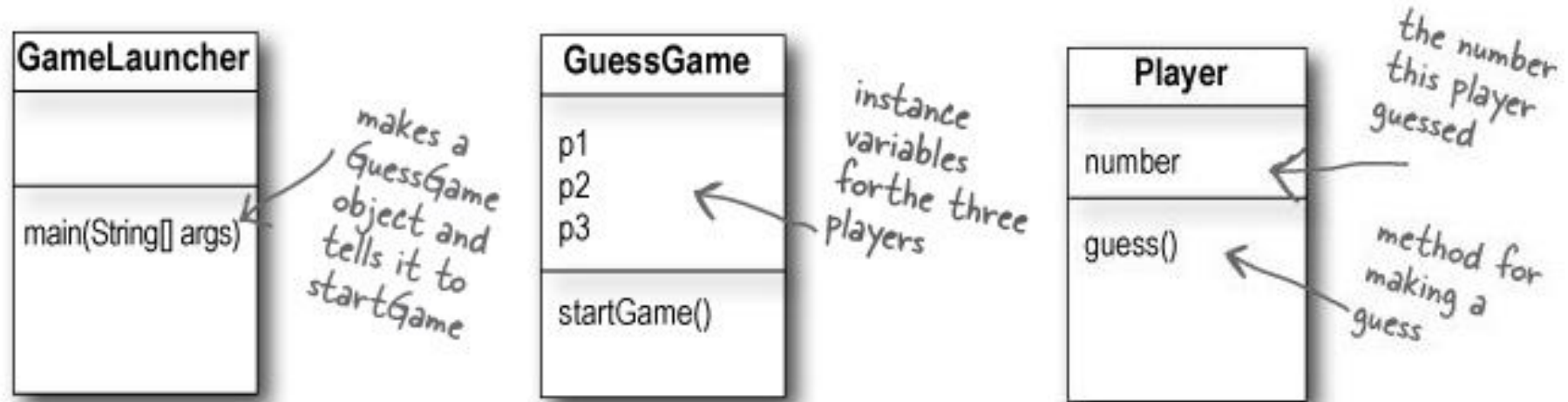
main의 두 가지 용도 :

실제 클래스를 테스트하기 위해

Java 애플리케이션을 시작하기 위해

A real Java application is nothing but objects talking to other objects. In this case, talking means objects calling methods on one another.

The Guessing Game



게임 요약:

- Guessing 게임에는 '게임' 객체와 3개의 '플레이어' 객체가 있다.
- 게임은 0에서 9 사이의 임의의 숫자를 생성하고, 3명의 선수 객체는 숫자를 추측한다. (절대적으로 재미있는 게임이라고 말하지 않았다.)
- 클래스: *GuessGame.class* *Player.class* *GameLauncher.class*

The Guessing Game

The Logic:

- 1) **GameLauncher** 클래스 \Rightarrow **Application**이 시작되는 곳
- 2) **main()** 메소드 \Rightarrow **GuessGame** 객체 생성, **startGame()** 메소드 호출
- 3) **GuessGame** 객체의 **startGame()** 메소드: 전체 게임이 펼쳐지는 곳.
 - ✓ 3명의 플레이어를 생성한 후, 임의의 숫자(플레이어가 추측할 대상)를 “생각한다”.
그런 다음 각 플레이어에게 추측을 요청하고 결과를 확인한 다음,
우승한 플레이어에 대한 정보를 표시하거나 다시 추측하도록 요청한다.

```
1 package com.skimok;
2
3 public class GuessGame {
4
5     Player p1;
6     Player p2;
7     Player p3;
8
9     public void startGame() {
10         p1 = new Player();
11         p2 = new Player();
12         p3 = new Player();
13         int guessp1 = 0;
14         int guessp2 = 0;
15         int guessp3 = 0;
16         boolean p1isRight = false;
17         boolean p2isRight = false;
18         boolean p3isRight = false;
19         int targetNumber = (int) (Math.random() * 10);
20         System.out.println("I'm thinking of a number between 0 and 9...");
21         while(true) {
22             System.out.println("Number to guess is " + targetNumber);
23
24             p1.guess();
25             p2.guess();
26             p3.guess();
27
28             guessp1 = p1.number;
29             System.out.println("Player one guessed " + guessp1);
30             guessp2 = p2.number;
```

```

31      System.out.println("Player two guessed " + guessp2);
32      guessp3 = p3.number;
33      System.out.println("Player three guessed " + guessp3);
34
35      if (guessp1 == targetNumber) {
36          plisRight = true;
37      }
38      if (guessp2 == targetNumber) {
39          p2isRight = true;
40      }
41      if (guessp3 == targetNumber) {
42          p3isRight = true;
43      }
44
45      if (plisRight || p2isRight || p3isRight)
46      {
47          System.out.println("We have a winner!");
48          System.out.println("Player one got it right? " + plisRight);
49          System.out.println("Player two got it right? " + p2isRight);
50          System.out.println("Player three got it right? " + p3isRight);
51          System.out.println("Game is over");
52          break;
53      }
54      else
55      {
56          System.out.println("Players will have to try again.");
57      }
58  }
59  }
60  }

```

Running the Guessing Game

Player.java

```
1  package com.skimok;
2
3  public class Player {
4      int number = 0;
5      public void guess()
6      {
7          number = (int) (Math.random() * 10);
8          System.out.println("I'm guessing " + number);
9      }
10 }
11
```

GameLauncher.java

```
1  package com.skimok;
2
3  public class GameLauncher {
4      public static void main (String[] args) {
5          GuessGame game = new GuessGame();
6          game.startGame();
7      }
8  }
9
```

실행 화면



```
run:
I'm thinking of a number between 0 and 9...
Number to guess is 7
I'm guessing 9
I'm guessing 8
I'm guessing 9
Player one guessed 9
Player two guessed 8
Player three guessed 9
Players will have to try again.
Number to guess is 7
I'm guessing 9
I'm guessing 4
I'm guessing 1
Player one guessed 9
Player two guessed 4
Player three guessed 1
Players will have to try again.
Number to guess is 7
I'm guessing 3
I'm guessing 1
I'm guessing 6
Player one guessed 3
Player two guessed 1
Player three guessed 6
Players will have to try again.
Number to guess is 7
I'm guessing 7
I'm guessing 8
I'm guessing 0
Player one guessed 7
Player two guessed 8
Player three guessed 0
We have a winner!
Player one got it right? true
Player two got it right? false
Player three got it right? false
Game is over
BUILD SUCCESSFUL (total time: 0 seconds)
```

실습과제 2-2 Guessing Game

본문의 **Guessing Game**을 구현하시오.

실습과제 2-3

```
File Edit Window Help Game
% java DrumKitTestDrive
bang bang ba-bang
ding ding da-ding
```

위와 같은 실행 결과가 나오도록 박스 안의 코드 조각들을 재배치하여 실행하시오.

```
d.playSnare();
```

```
DrumKit d = new DrumKit();
```

```
boolean topHat = true;
boolean snare = true;
```

```
void playSnare() {
    System.out.println("bang bang ba-bang");
}
```

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

```
if (d.snare == true) {
    d.playSnare();
}
```

```
d.snare = false;
```

```
class DrumKitTestDrive {
```

```
d.playTopHat();
```

```
class DrumKit {
```

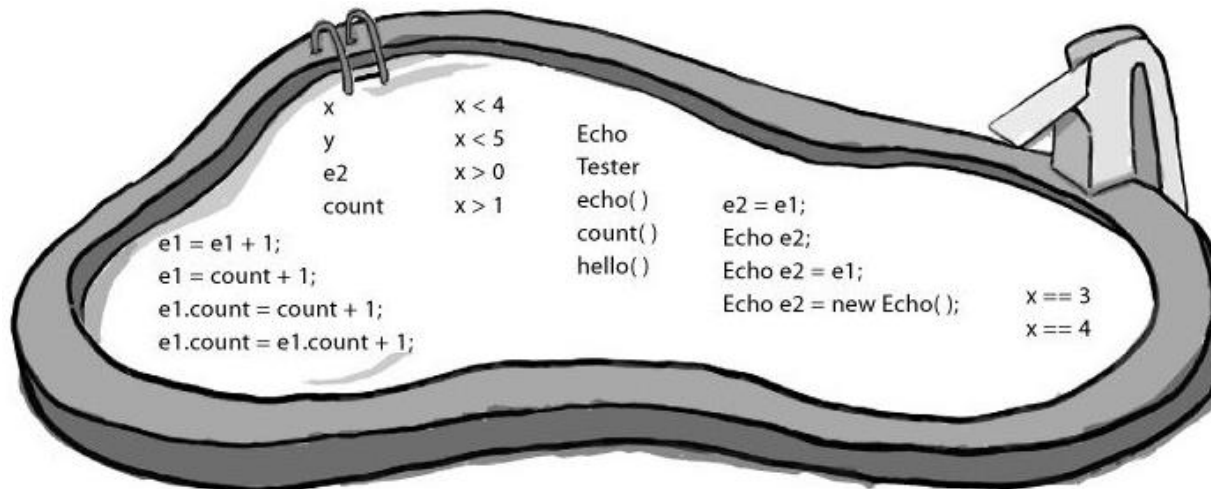
```
void playTopHat () {
    System.out.println("ding ding da-ding");
}
```

실습과제 2-4 Pool Puzzle

Pool의 코드 조각을 1회 이상 사용할 수 있다.

Output

```
File Edit Window Help Implode
%java EchoTestDrive
helloooo...
helloooo...
helloooo...
helloooo...
10
```



```
public class EchoTestDrive {
    public static void main(String [] args) {
        Echo e1 = new Echo();

        int x = 0;
        while ( _____ ) {
            e1.hello();

            if ( _____ ) {
                e2.count = e2.count + 1;
            }
            if ( _____ ) {
                e2.count = e2.count + e1.count;
            }
            x = x + 1;
        }
        System.out.println(e2.count);
    }
}
```

```
class _____ {
    int _____ = 0;
    void _____ {
        System.out.println("helloooo... ");
    }
}
```

[Project 1: Word Search Game]

Due: 03/17

사이버캠퍼스 '과제' 참고!!

Submission Instructions:

Please upload copies of your program source code (*.java) and two or more sample runs for **Project 1: Word Search Game**. For the source code, you may simply compress (i.e., zip) your entire project folder (NetBeans, Eclipse, etc.) and upload that to Canvas instead of uploading the individual Java source code files. For the sample runs, copy and paste the output for two complete runs of the program into *MS Word*, then save the Word document. You must also submit printouts of your source code and sample run files, stapled together, with your name on the top sheet. Printouts are required if you want your submission to be considered for a grade.

PLEASE NOTE THAT PICTURE FILES (e.g., *.png, *.jpg) NOT PASTED INTO WORD DOCUMENTS WILL NOT BE ACCEPTED.

