9장. 객체의 삶과 죽음

스택과 힙
지역변수와 인스턴스 변수
객체 생성과 생성자
객체 제거 (가비지 컬렉션)

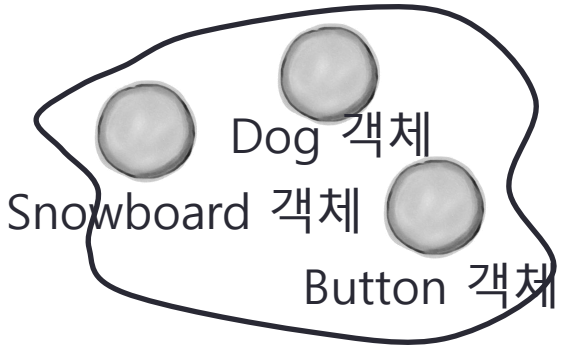
객체의 삶과 죽음


그리고 그가 말했어. “다리에 감각이 없어!”
그리고 내가 말했지. “조! 정신 차려 조!” 하
지만 이미 너무 늦었어. 가비지 컬렉터가
나타났고 그는 죽고 말았지. 내가 만나 본
가장 좋은 객체였는데 말야…

스택과 힙

- 스택 (stack)
  - 메서드 호출과 지역 변수가 사는 곳
  - 지역 변수는 스택 변수라고도 부릅니다.

- 힙 (heap)
  - 모든 객체가 사는 곳
  - 인스턴스 변수는 객체 안에 들어있습니다.



인스턴스 변수와 지역 변수

- **인스턴스 변수** (instance variable)
  - 클래스 내에서 선언한 변수
  - 그 변수가 속한 객체 안에서 삽니다.

```
public class Duck {
  int size;
}
```

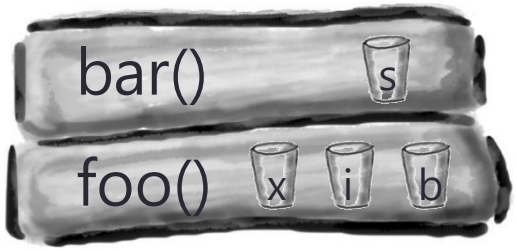
- **지역 변수** (local variable)
  - 메서드 내에서 선언한 변수
  - **매개변수**도 지역 변수에 포함됩니다.

```
public void foo(int x) {
  int i = x + 3;
  boolean b = true;
}
```

p.280

스택과 메서드

- 메서드를 호출하면 메서드는 호출 스택(call stack) 맨 위에 올라갑니다.
- 스택 맨 위에 있는 메서드는 그 스택에서 현재 실행중인 메서드입니다.
- 메서드는 그 끝을 나타내는 중괄호에 다다를 때까지 스택에 머무릅니다.
- foo()라는 메서드에서 bar()라는 메서드를 호출하면 bar() 메서드가 foo() 위에 얹힙니다.



p.281

스택과 메서드

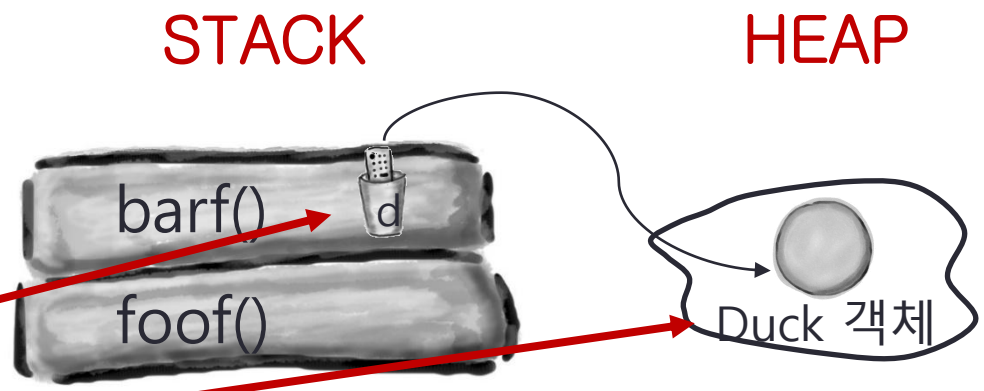
```
public void doStuff() {
  boolean b = true;
  go(4);
}

public void go(int x) {
  int z = x + 24;
  crazy();
}

public void crazy() {
  char c = 'a';
}
```

p.281

지역 변수로 쓰이는 객체는?

- 원시 변수가 아닌 변수에는 객체에 대한 레퍼런스가 들어있습니다.
- 지역 변수가 객체 레퍼런스인 경우에는 변수(레퍼런스)만 스택에 들어갑니다.
- 객체 자체는 여전히 힙 안에 들어있습니다.

```
public class StackRef {
  public void foof() {
    barf();
  }
  public void barf() {
    Duck d = new Duck(24);
  }
}
```



p.282

바보 같은 질문은 없습니다

- 이런 내용을 왜 배우는 거죠?
  - 변수 영역, 객체 생성 문제, 메모리 관리, 스레드, 예외 처리 등을 이해하는 데 있어서 스택과 힙에 관한 내용은 필수적입니다.
  - 특정 JVM, 플랫폼에서 스택과 힙을 구현하는 방법은 몰라도 됩니다.
  - 일단 스택과 힙에 대해 이해하고 나면 다른 내용을 이해하는 것이 한결 수월해집니다.
  - (참고)
    - 소프트웨어 개발 보안 가이드, https://www.kisa.or.kr/2060204/form?postSeq=5&page=1
    - SEI CERT Oracle Coding Standard for Java, https://wiki.sei.cmu.edu/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java
      → OBJ05-J. Do not return references to private mutable class members

```
public Date getDate() {
  return (Date)d.clone();
}
```

p.282

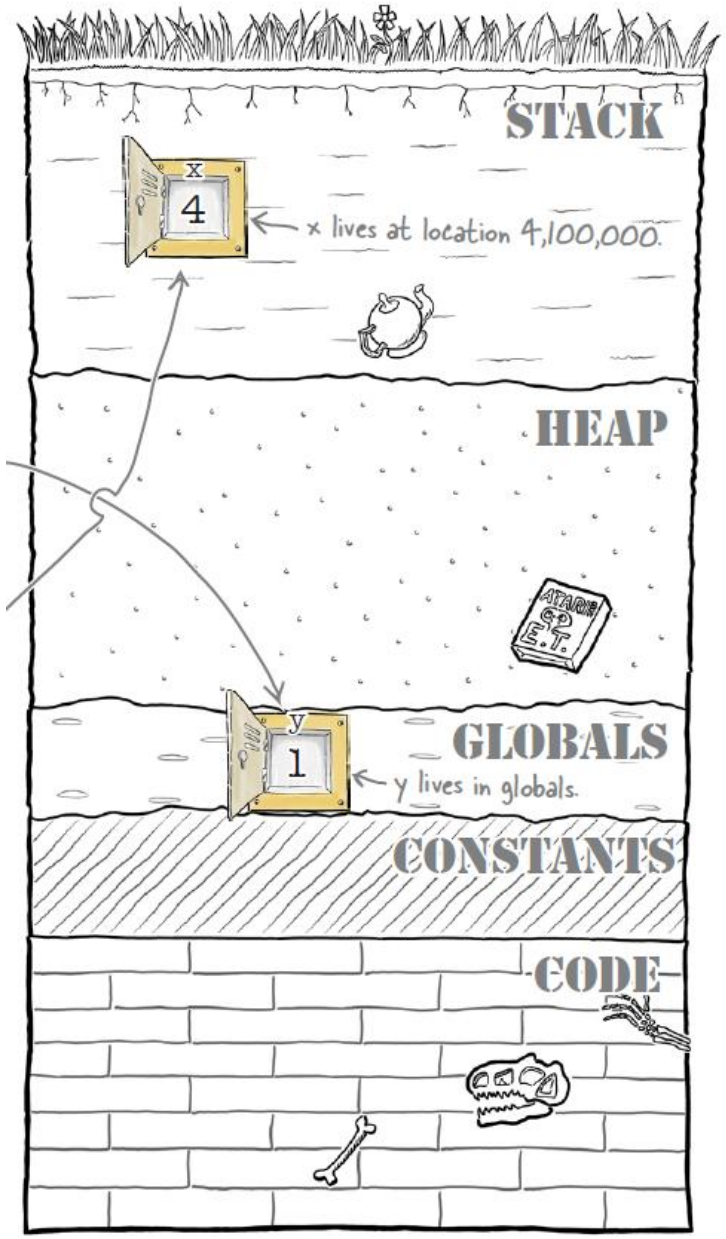
핵심 정리

- 자바에서 우리가 관심을 가져야 할 메모리 공간에는 **힙(heap)**과 **스택**, 이렇게 두 가지가 있습니다.
- 클래스 안에서, 하지만 메서드 밖에서 선언된 변수는 **인스턴스 변수**입니다.
- 메서드 안에서 선언된 변수, 또는 **매개변수**는 지역 변수입니다.
- 모든 **지역 변수**는 스택에 들어있으며 그 변수를 선언한 메서드에 해당하는 프레임 안에 들어있습니다.
- 객체 레퍼런스 변수도 지역 변수라면 스택에 저장됩니다.
- 모든 객체는 힙에 저장됩니다.

C언어 메모리 구조

- 참고: Head First C

- C 언어에는 전역 변수 개념도 있음.



JVM 메모리 구조

- 참고 URL: https://www.yourkit.com/docs/kb/sizes.jsp

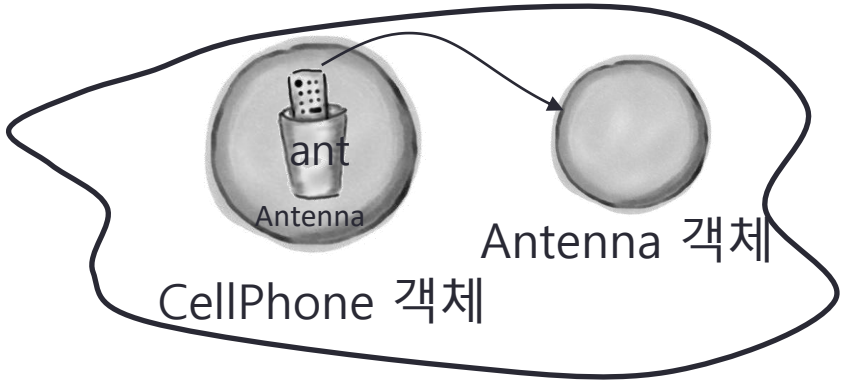
- Heap 메모리
  - 모든 클래스 인스턴스와 배열이 할당되는 런타임 데이터 영역
  - -Xmx<size>: 최대 힙 크기, -Xms<size>: 초기 힙 크기

- Non-Heap 메모리
  - JVM 시작 시 만들어져서 클래스 별 구조(런타임 상수 풀, 필드, 메서드 데이터, 생성자와 메서드 코드)를 저장

인스턴스 변수는 어디에?


```
public class CellPhone {
  private Antenna ant;
} // 초기화하지 않은 경우
```


```
public class CellPhone {
  private Antenna ant = new Antenna();
} // 초기화한 경우
```



p.283

객체 선언, 생성, 대입

Duck myDuck = new Duck();

1. 레퍼런스 변수 선언

Duck myDuck = new Duck();

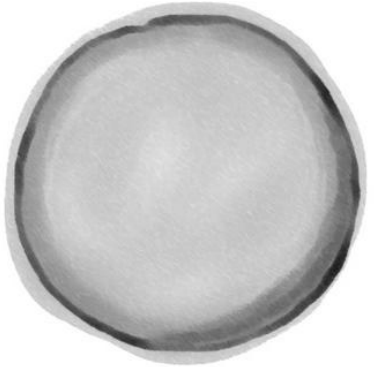
Duck 레퍼런스

객체 선언, 생성, 대입

```
Duck myDuck = new Duck();
```

2. 객체 생성

```
Duck myDuck = new Duck();
```



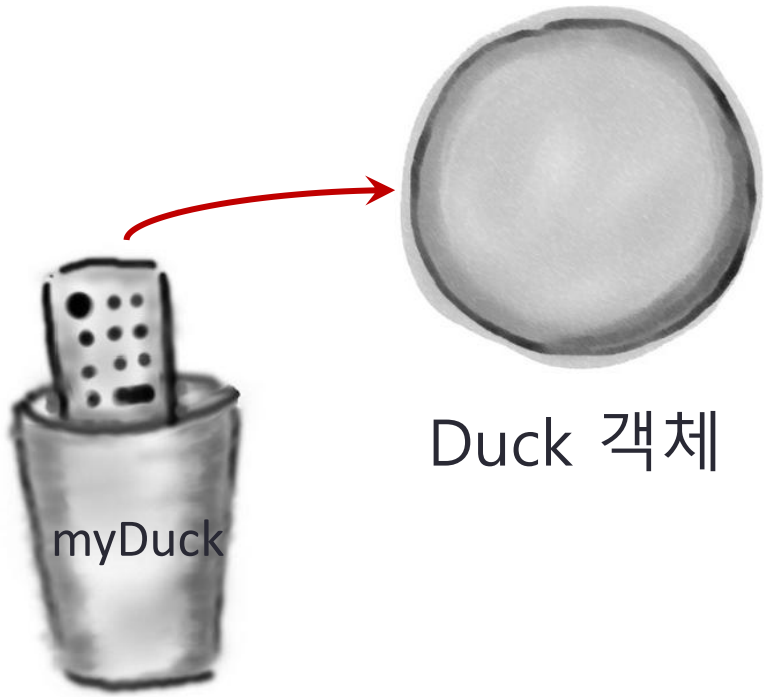
Duck 객체

객체 선언, 생성, 대입

Duck myDuck = new Duck();

3. 객체와 레퍼런스 연결

Duck myDuck = new Duck();



메서드? 생성자!

- 생성자 (Constructor)
  - 객체를 생성할 때 실행되는 코드가 들어있습니다.
  - 모든 클래스에 생성자가 있습니다.
  - 직접 만들지 않아도 컴파일러에서 자동으로 생성자를 만들어줍니다.

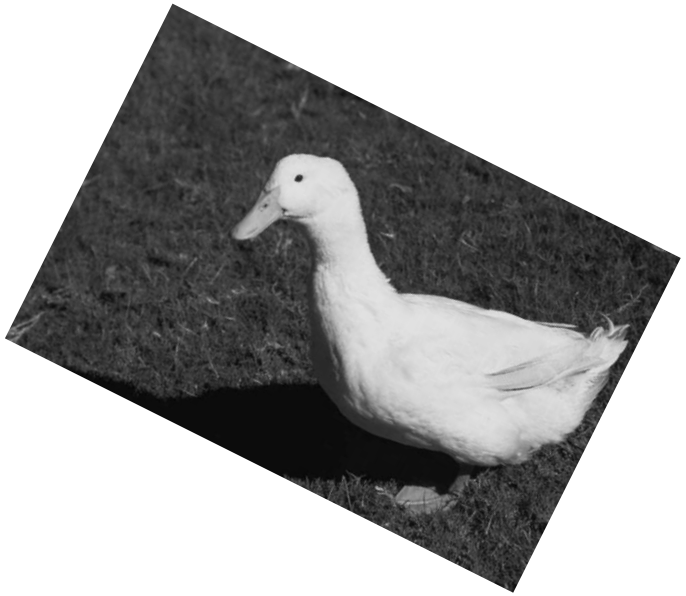
```
Duck myDuck = new Duck();
```

```
public Duck() {
  // 생성자 코드가 들어갈 자리
}
```

p.285

생성자 예제

```
public class Duck {
  public Duck() {
    System.out.println("Quack");
  }
}

public class UseADuck {
  public static void main(String[] args) {
    Duck d = new Duck();
  }
}
```



```
$ java UseADuck
Quack
```

p.286

객체의 상태를 초기화하는 방법

- 객체의 상태를 초기화하는 작업은 대부분 생성자에서 처리합니다.

```
public Duck() {
    size = 34;
}
```

- Duck을 사용하는 프로그래머가 오리의 크기를 결정하도록 하려면?

```
public class Duck {
    int size;                                   ← instance variable

    public Duck() {
        System.out.println("Quack");           ← constructor
    }

    public void setSize(int newSize) {          ← setter method
        size = newSize;
    }
}

public class UseADuck {

    public static void main (String[] args){
        Duck d = new Duck();
        d.setSize(42);
    }
}
```

There's a bad thing here. The Duck is alive at this point in the code, but without a size!* And then you're relying on the Duck-user to KNOW that Duck creation is a two-part process: one to call the constructor and one to call the setter.

*Instance variables do have a default value. 0 or 0.0 for numeric primitives, false for booleans, and null for references.

p.287

바보 같은 질문은 없습니다

- 컴파일러에서 자동으로 만들어주는데 왜 생성자를 따로 만들어야 하나요?
  - 컴파일러에서 자동으로 만들어 주는 생성자: **기본 생성자**
    - 매개변수 X, 생성자 내 코드 X → 아무 일도 안 함. 단지 객체 생성에만 사용됨.

  - **객체 초기화 작업 및 각종 준비 작업이 필요하다면 생성자를 따로 만들어야** 합니다.
  - 객체를 완전히 만들기 전에 사용자로부터 뭔가를 입력 받는다거나 하는 경우에는 생성자가 반드시 필요합니다.
  - 상위 클래스 생성자 문제로 인해 초기화 및 준비 작업이 필요하지 않은 경우에도 생성자를 만들어야 할 수도 있습니다.

  - (LJM) **생성자에서 반드시 해야 하는 일: 인스턴스 변수(또는 필드)의 초기화**
    → 인스턴스가 만들어진 후 항상 같은 방법으로 상태 초기화되어야만 메서드 호출을 통해 일을 시킬 때 동일한 동작 보장 가능

p.287

바보 같은 질문은 없습니다

$ java Duck.java 로 실습해 볼 것!

- 메서드와 생성자를 어떻게 구분할 수 있나요? 그리고 클래스와 이름이 같은 메서드를 만들 수 있나요?
  - 가능합니다. 클래스와 같은 이름을 가지고 있다고 해서 무조건 생성자가 되는 것은 아닙니다.
  - 생성자와 메서드는 리턴 유형 유무로 구분합니다. **생성자에는 리턴 유형(또는 반환 자료형)이 없어야** 하고 메서드에는 리턴 유형이 있어야만 합니다.

```
public Duck() { ... }
public int Duck() { ... }
```

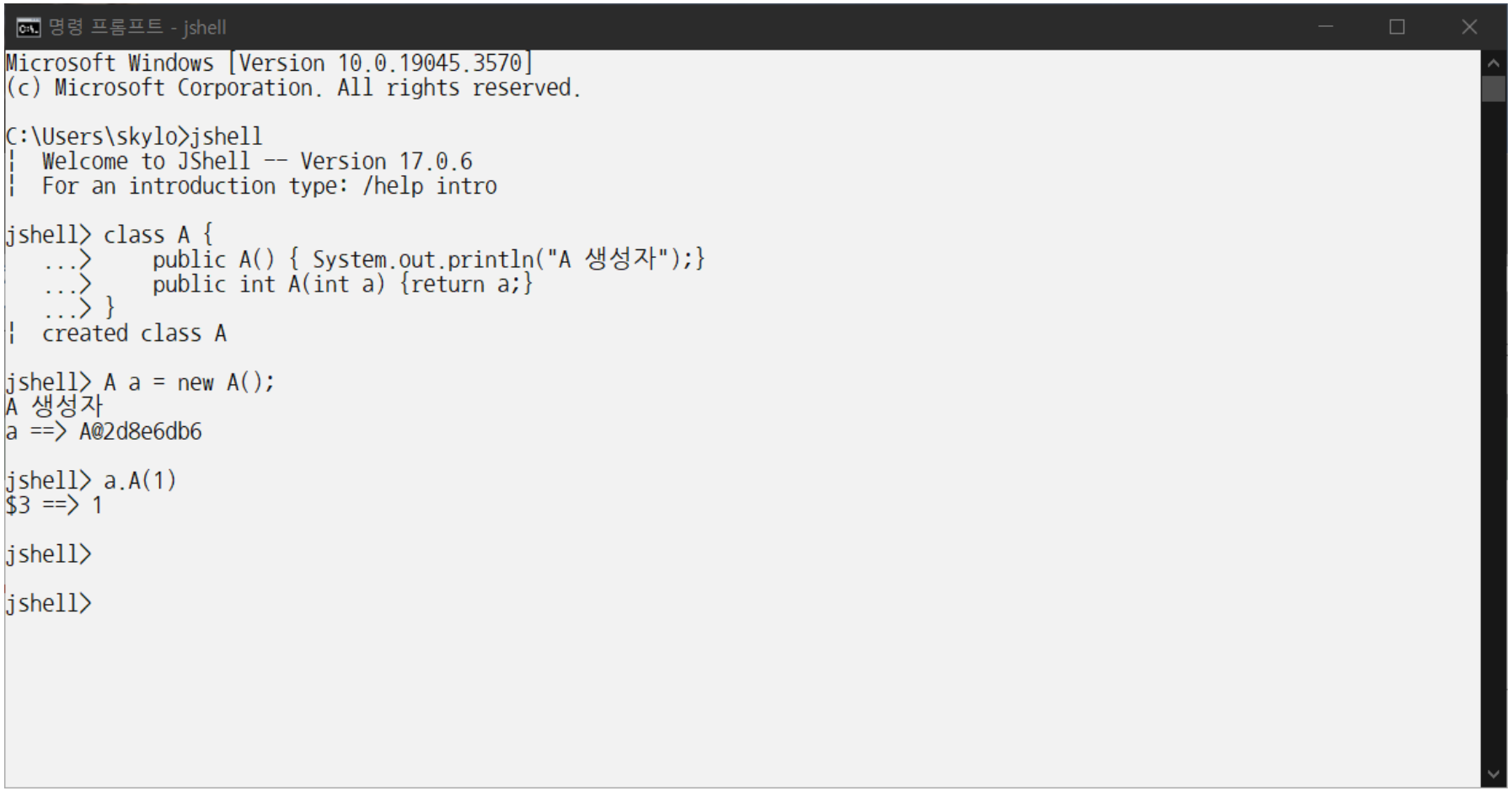
```
6   package cse.oop2.ch09.p277;
7
8   /**...4 lines */
12  public class Duck {
13
14      public Duck() {  // 생성자: 반환 값의 자료형 없음!
15          System.out.println("기본 생성자 호출");
16      }
17
18      public int Duck() {  // 메소드: int형 반환
19          return 10;
20      }
21
22      /**...3 lines */
25      public static void main(String[] args) {
26          Duck d = new Duck();
27          System.out.println("return value = " + d.Duck());
28      }
29
30  }
```

```
--- exec-maven-plug
기본 생성자 호출
return value = 10
```

p.287

따라 해 봅시다.

- 클래스 A의 생성자 A()와 메서드 A(int a): int



바보 같은 질문은 없습니다

- 생성자도 상속되나요? 상위클래스에서만 생성자를 만들고 하위클래스에서 생성자를 만들지 않으면 기본 생성자 대신 상위클래스의 생성자가 쓰이나요?
  - 아닙니다. **생성자는 상속되지 않습니다.**
  - 이와 관련된 내용은 잠시 후에 살펴보겠습니다.

p.287

생성자를 이용한 초기화

- 인스턴스 변수가 초기화되기 전까지 객체를 사용해선 안 된다면 **생성자에서 초기화**하면 됩니다.

```
public class Duck {
  int size;
  public Duck(int duckSize) {
    System.out.println("Quack");
    size = duckSize;
    System.out.println("size is " + size);
  }
}

public class UseADuck {
  public static void main(String[] args) {
    Duck d = new Duck(42);
  }
}
```

```
$ java UseADuck
Quack
size is 42
```

p.288

인자 (형식) 매개변수가 없는 생성자

- 형식 매개변수(formal parameter)가 있는 생성자만 있으면 사용하기가 불편합니다.
- 기본 크기가 자동으로 정해지는 인자가 없는 Duck 생성자를 만들면 편하지 않을까요?

```
public class Duck {
  int size;
  public Duck(int newSize) {
    if (newSize == 0) {
      size = 27;
    } else {
      size = newSize;
    }
  }
}
```

```
public class Duck2 {
  int size;
  public Duck2() {
    size = 27;  // or this(27);
  }
  public Duck2(int duckSize) {
    size = duckSize;
  }
}
```

if문은 복잡도 증가
→ McCabe Complexity

```
Duck2 d = new Duck2(15);
Duck2 d = new Duck2();
```

p.289

인자가 없는 생성자

- 인자가 없는 생성자는 컴파일러에서 자동으로 만들어주지 않나요?
  - 아닙니다. 컴파일러에서는 생성자가 전혀 없는 경우에만 생성자를 자동으로 생성함.
  - 인자가 있는 생성자를 만들었을 때 인자가 없는 생성자도 필요하다면 인자가 없는 생성자도 직접 생성
  - 한 클래스에 생성자가 두 개 이상 있으면 각 생성자의 인자 목록은 반드시 서로 달라야 함.

p.290

생성자 오버로딩

- 두 개 이상의 생성자가 필요하다면 생성자 오버로딩을 사용하면 됩니다.
- 각 생성자의 인자 목록은 서로 달라야만 합니다.

```
Public class Mushroom {
   public Mushroom(int size) { }
   public Mushroom( ) { }
   public Mushroom(boolean isMagic) { }
   public Mushroom(boolean isMagic, int size) { }
   public Mushroom(int size, boolean isMagic) { }
}
```

public Mushroom(int height) 가능?

p.291

핵심 정리

- 인스턴스 변수는 그 변수가 들어있는 객체 안에 저장됩니다.
- 인스턴스 변수가 레퍼런스 변수인 경우에는 레퍼런스와 객체가 모두 힙에 저장됩니다.
- new 키워드를 사용할 때 실행되는 코드를 생성자라고 합니다.
- 생성자명은 반드시 클래스명과 같아야 하며 리턴 유형은 없어야 합니다.
- 생성자를 이용하여 객체의 상태(인스턴스 변수)를 초기화할 수 있습니다.
- 클래스에 생성자가 없으면 컴파일러에서 기본 생성자를 생성
  → public ClassName() { } // does nothing
- 기본 생성자에는 인자가 없습니다.
- 생성자를 하나라도 만들면 컴파일러에서 기본 생성자를 만들어주지 않습니다.

핵심 정리

- 인자가 없는 생성자를 만들고 싶은데 인자가 있는 생성자가 따로 있다면 인자가 없는 생성자도 직접 만들어야 합니다.
- 가능하면 인자가 없는 생성자도 만드는 것이 좋습니다.
- 생성자 오버로딩을 활용하면 한 클래스에 두 개 이상의 생성자를 만들 수 있습니다.
- 오버로딩된 생성자들의 인자 목록은 반드시 서로 달라야 합니다.
- 인자 목록이 똑같은 생성자가 두 개 이상 있을 수 없습니다.
- 인스턴스 변수에는 자동으로 기본값이 지정됩니다. **원시 유형의 기본값은 0/0.0/false**이며 **객체에 대한 레퍼런스의 기본값은 null**입니다.

바보 같은 질문은 없습니다

- 기본값을 지정할 수 없기 때문에 인자가 없는 생성자를 만들지 않아야 하는 경우는 없나요?
  - 물론 적당한 기본값이 없는 경우에는 인자가 없는 생성자를 만드는 것이 무의미할 수도 있습니다.
  - 예) Color 클래스

```
Color c = new Color(3, 45, 200);
~~Color c = new Color();~~
```

```
cannot resolve symbol
: constructor Color()
location: class java.awt.Color
Color c = new Color();
          ^
1 error
```

생성자에 대해 반드시 알아야 할 네 가지

1. 생성자는 누군가가 어떤 클래스 유형에 대해 new를 쓸 때 실행되는 코드입니다.

```
Duck d = new Duck();
```

2. 생성자명은 반드시 클래스명과 같아야 하며 리턴 유형은 없습니다.

```
public Duck(int size) { }
```

3. 클래스를 만들 때 생성자를 만들지 않으면 컴파일러에서 기본 생성자를 자동으로 추가해 줍니다. 기본 생성자는 언제나 인자가 없는 생성자입니다.

```
public Duck() { }
```

p.293

생성자에 대해 반드시 알아야 할 네 가지

4. 인자 목록만 다르면 한 클래스에 생성자를 여러 개 만들 수도 있습니다. 한 클래스에 두 개 이상의 생성자가 있으면 오버로드된 생성자가 있다고 말합니다.

```
public Duck() { }
public Duck(int size) { }
public Duck(String name) { }
public Duck(String name, int size) { }
public Duck(int size, String name) { }
```

→ Telescoping constructor pattern (점층적 생성자 패턴)

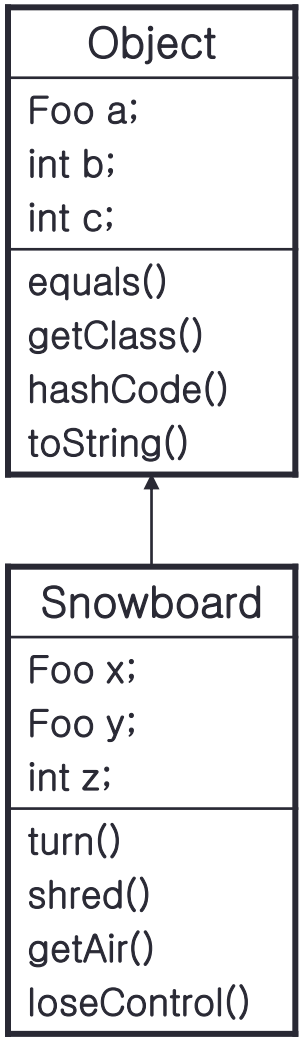
→ 대안: Builder pattern

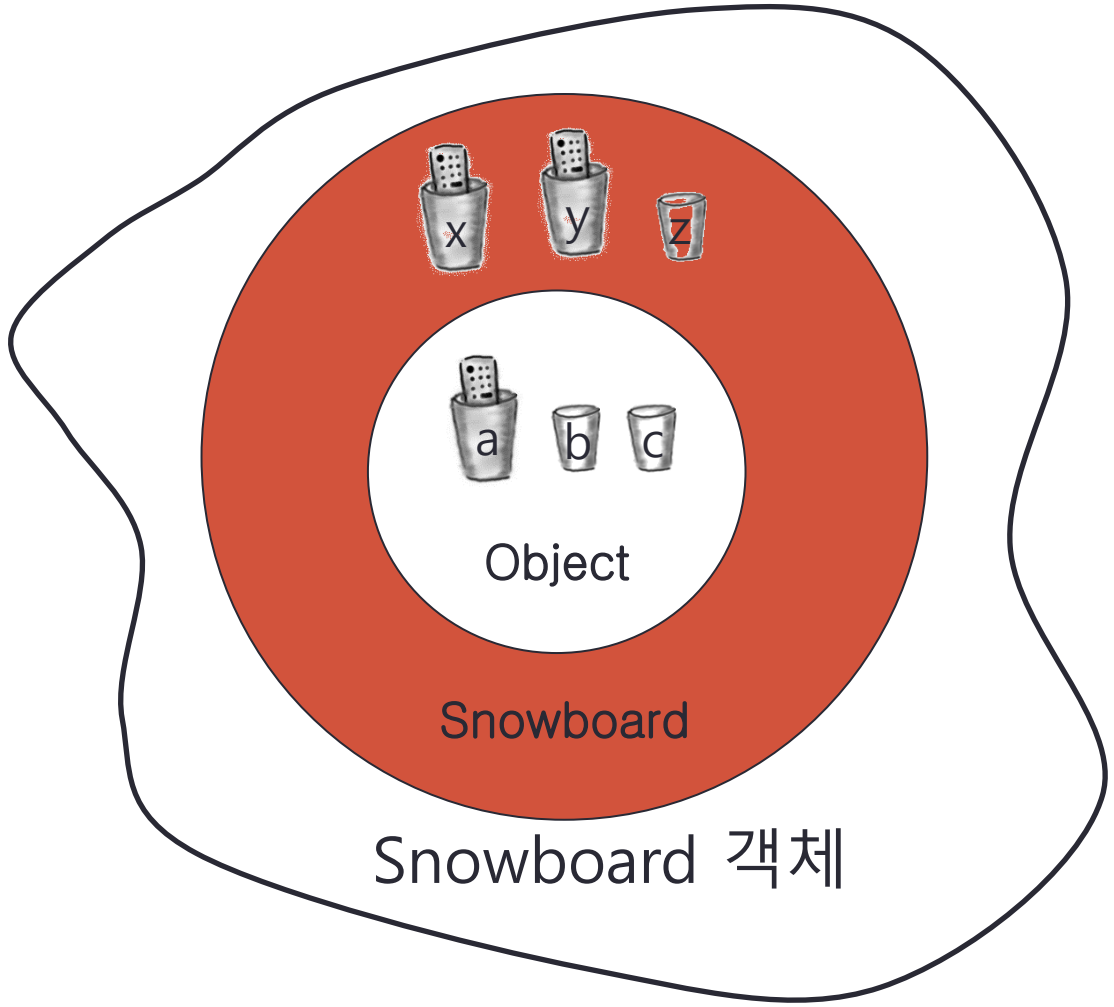
바보 같은 질문은 없습니다

- 생성자는 반드시 public이어야 하나요?
  - 아닙니다. 생성자도 public, private, default로 지정할 수 있습니다.

- private 생성자는 어떤 용도로 쓰나요? 아무도 그 생성자를 호출할 수 없으면 그 생성자를 가지고 새로운 객체를 만들 수 없지 않나요?
  - 그렇진 않습니다. 클래스 "밖에서" 접근할 수 없을 뿐입니다. 즉 같은 클래스 안에 있는 코드에서는 그 생성자를 사용할 수 있습니다.
  - 대표적인 사용 사례: **싱글턴 패턴(singleton pattern)**

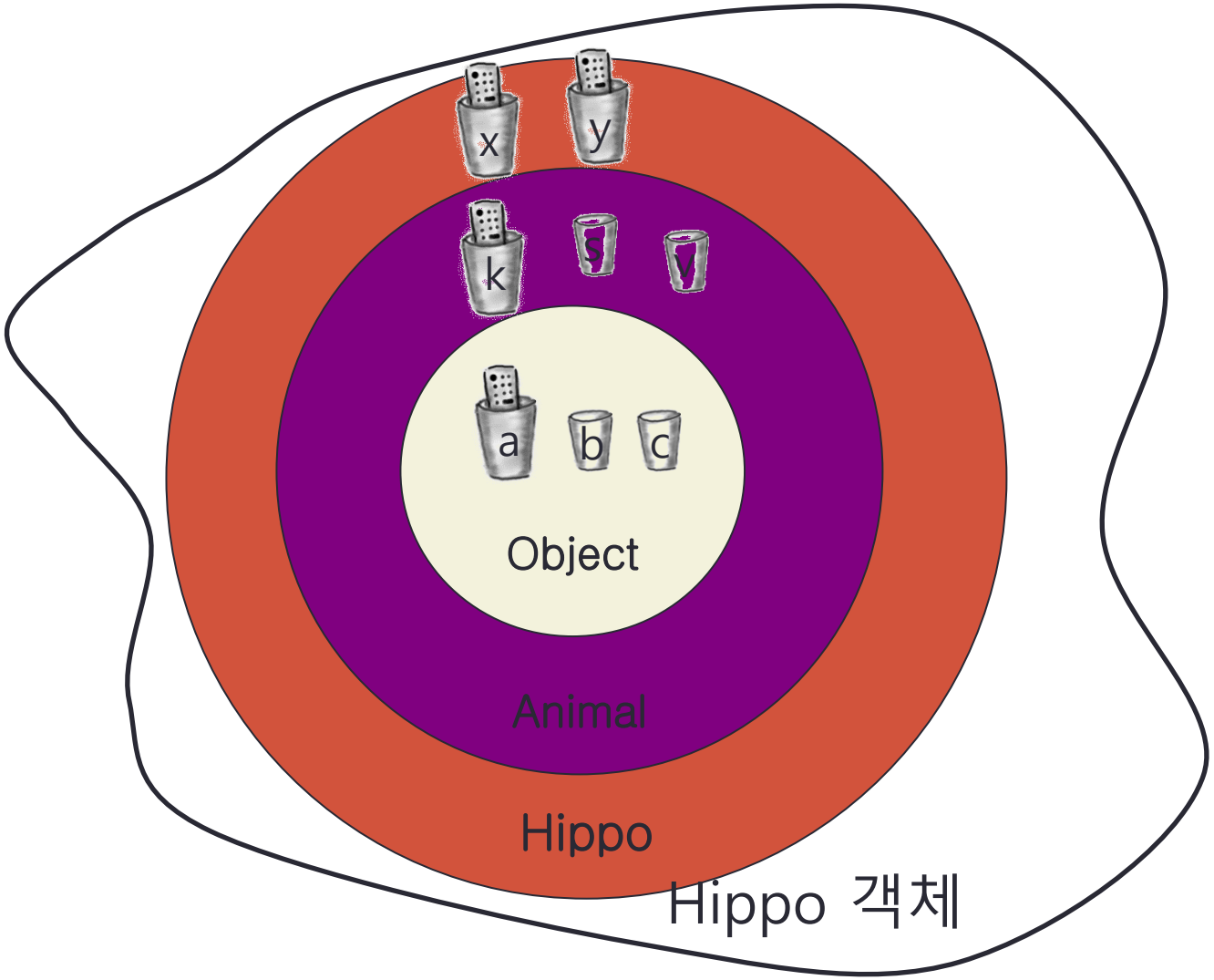
상위클래스와 상속, 생성자 사이의 관계

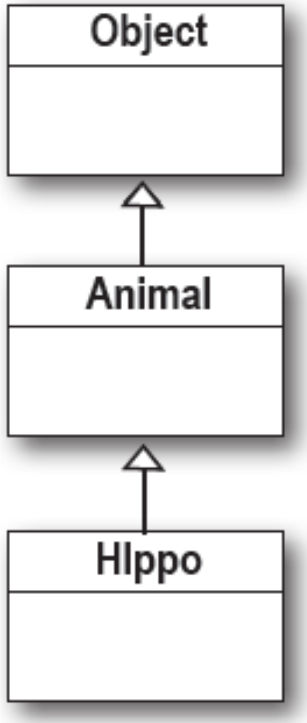




p.294

상위클래스 생성자의 역할





생성자 연쇄
(constructor chaining)

p.295

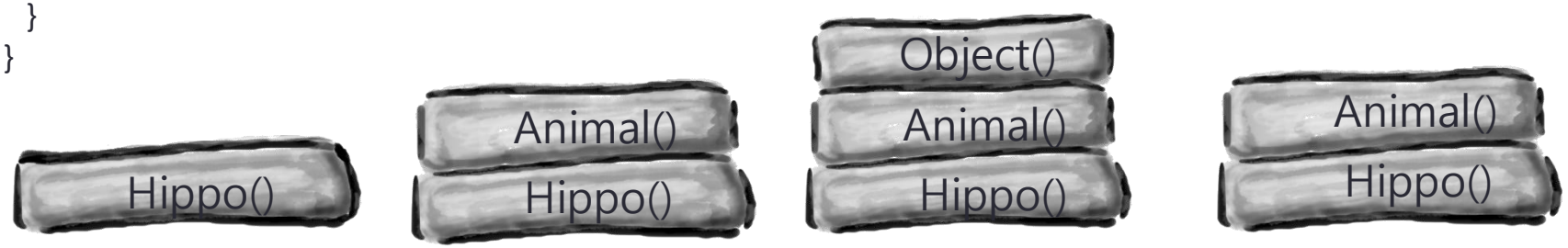
생성자 연쇄

```
public abstract class Animal {
  public Animal() {
    System.out.println("Making an Animal");
  }
}

public class Hippo extends Animal {
  public Hippo() {
    System.out.println("Making a Hippo");
  }
}

public class TestHippo {
  public static void main(String[] args) {
    System.out.println("Starting...");
    Hippo h = new Hippo();
  }
}
```

어떤 생성자가 먼저 호출?

```
$ java TestHippo
Starting...
Making an Animal
Making a Hippo
```

```
--- exec-maven-plugin:
Starting...
Making an Animal
Making a Hippo
```



상위클래스 생성자 호출 방법

```
public class Duck extends Animal {
   int size;
   public Duck(int newSize) {
      Animal();
      size = newSize;
   }
}
```

```
public class Duck extends Animal {
   int size;
   public Duck(int newSize) {
      super();
      size = newSize;
   }
}
```

p.297

상위클래스 생성자

1. 생성자를 만들지 않은 경우
   → 컴파일러에서 다음과 같은 내용을 추가합니다.

```
public ClassName() {
  super();
}
```

- 생성자를 만들긴 했는데 super()를 호출하지 않은 경우
  → 컴파일러에서 super()를 자동으로 추가해줍니다.

```
Public MyClass() {
    // 필드 초기화
}
```

```
Public MyClass() {
    super();
    // 필드 초기화
}
```

p.297

super() 호출 선언문의 위치

- 자식이 있으려면 반드시 부모가 먼저 있어야 합니다.
  - 마찬가지로 상위클래스 생성이 끝나야만 하위클래스 생성이 끝날 수 있습니다.
  - super()를 호출하는 선언문은 모든 생성자의 첫 번째 선언문이어야 합니다.

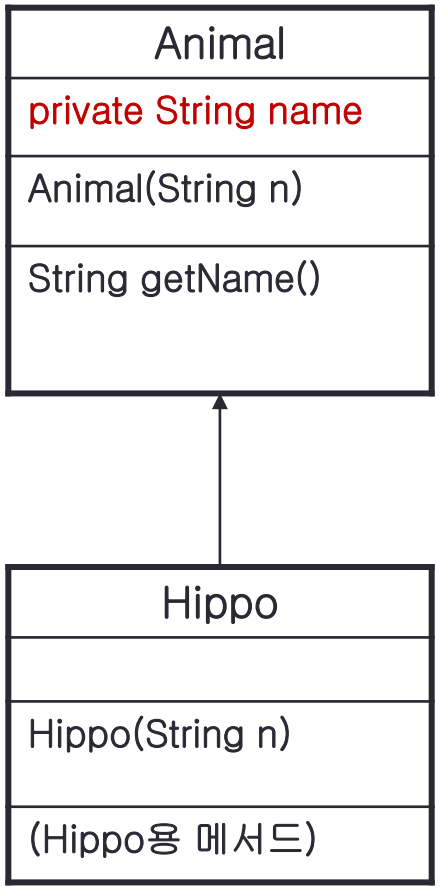
```
public Boop() {
  super();
}

public Boop(int i) {
  super();
  size = i;
}
```

```
public Boop() {
}

public Boop(int i) {
  size = i;
}

public Boop(int i) {
  size = i;
  super();  // 컴파일러 오류
}
```

p.298

인자가 있는 상위클래스 생성자

```
public abstract class Animal {
  private String name;

  public String getName() {
    return name;
  }

  public Animal(String theName) {
    name = theName;
  }
}

public class Hippo extends Animal {
  public Hippo(String name) {
    super(name); // Animal 필드 name 초기화
  }
}

public class MakeHippo {
  public static void main(String[] args) {
    Hippo h = new Hippo("Buffy");
    System.out.println(h.getName());
  }
}
```




```
$ java MakeHippo
Buffy
```

p.299

오버로딩된 생성자 호출 방법

- 같은 클래스에 있는 다른 생성자를 호출할 때는 **this()**를 쓰면 됩니다.
- **this()는 생성자에서만 호출**할 수 있으며 반드시 그 **생성자의 첫 번째 선언문**이어야만 합니다.
- **super()와 this()를 동시에 호출할 수는 없습니다**.

```
class Mini extends Car {
  Color color;
  public Mini() {
    this(Color.Red);
  }
  public Mini(Color c) {
    super("Mini");
    color = c;
  }
  public Mini(int size) {
    this(Color.Red);
    super(size);
  }
}
```

p.300

```
$ javac Mini.java
Mini.java:16: call to super must be first
statement in constructor
  super();
      ^
```

객체의 생존 기간

- 지역 변수
  - 그 변수를 선언한 메서드 안에서만 살 수 있음

```
public void read() {
  int s = 42;
}
```

- 인스턴스 변수
  - 객체가 살아있는 동안 살 수 있음

```
public class Life {
  int size;
  public void setSize(int s) {
    size = s;
  }
}
```

```
{
    Life myLife = new Life();  size


}
```

p.302

지역 변수의 삶과 영역

- 삶(life)
  - 지역 변수는 스택 프레임이 스택에 들어있는 한 계속 살아있습니다.
- 영역 → **범위 (scope)**
  - 지역 변수의 영역은 그 변수를 선언한 메서드 내로 제한됩니다.
  - **변수 정의한 문장부터 정의된 블록({ … })을 벗어날 때까지**


```
public void doStuff() {
  boolean b = true;
  go(4);
}

public void go(int x) {
  int z = x + 24;
  crazy();
}

public void crazy() {
  char c = 'a';
}
```

p.303

레퍼런스 변수의 삶

- 마지막 레퍼런스가 사라지면 그 객체는 가비지 컬렉션 대상이 됩니다.
- 객체 레퍼런스 제거 방법

```
void go() {
   Life z = new Life();
}
```

레퍼런스가 영역을 벗어남

```
Life z = new Life();
z = new Life();
```

다른 객체 대입

```
Life z = new Life();
z = null;
```

레퍼런스를 null로 설정

p.304

숙제

- 본문을 꼼꼼하게 읽어보세요.
- 본문에 들어있는 연필을 깎읍시다, 두뇌 운동 등을 전부 여러분 힘으로 해결해보세요.
- 연습문제를 모두 풀어보세요. 퍼즐도 해 보는 것이 좋습니다.

JLS11 vs. Head-First Java

8.8.9 Default Constructor

If a class contains no constructor declarations, then a default constructor is implicitly declared. The form of the default constructor for a top level class, member class, or local class is as follows:

- The default constructor has the same access modifier as the class, unless the class lacks an access modifier, in which case the default constructor has package access (§6.6).
- The default constructor has no formal parameters, except in a non-`private` inner member class, where the default constructor implicitly declares one formal parameter representing the immediately enclosing instance of the class (§8.8.1, §15.9.2, §15.9.3).

Make it easy to make a Duck

Be sure you have a no-arg constructor

What happens if the Duck constructor takes an argument? Think about it. On the previous page, there's only *one* Duck constructor—and it takes an int argument for the *size* of the Duck. That might not be a big problem, but it does make it

- (참고) https://stackoverflow.com/questions/156767/whats-the-difference-between-an-argument-and-a-parameter

- (이전 페이지로 돌아가기)