비타민 퀴즈 정답

변수 선언의 위치

chapter 02

5행에 있는 abs 변수의 선언문을 if 조건문 바로 앞으로 이동시키면 됩니다.

유별난 final 변수

chapter 02

복합 대입 연산자, 증가 연산자, 감소 연산자는 모두 변수의 기존 값을 이용하여 계산을 수행한 후, 그 결과를 다시 원래의 변수에 대입하는 연산자입니다. 그런데 final 변수에는 한번 값을 대입한 다음에 새로운 값을 대입할 수 없습니다. 그래서 이런 연산자들을 사용할 수 없는 겁니다.

else if 절과 switch 문

chapter 02

[예제 2-26]의 if 문과 똑같은 기능을 하는 switch 문을 만들기 위해서는 너무나 많은 case 문이 필요합니다. 그렇기 때문에 이 경우에는 switch 문보다는 if 문을 사용하는 것이 적당합니다.

while 문과 do-while 문의 비교

chapter 02

WhileExample은 cnt 변수 값을 하나도 출력하지 않지만, DoWhileExample은 cnt 값이 10일 때 그 값을 출력합니다. do-while 문은 조건식을 검사하기 전에 무조건 한 번 반복 부분을 실행하기 때문입니다.

향상된 for 문

chapter 02

첫 번째 for 문은 배열 항목의 값을 가져다가 사용만 하는 것이 아니라 다른 값으로 바꾸고 있습니다. 그렇기 때문에 이 for 문은 향상된 for 문 형식으로 고칠 수 없습니다. 그렇지만 두 번째 for 문은 배열 항목의 값을 가져다가 사용만 하기 때문에 향상된 for 문 형식으로 고칠 수 있습니다.

2진 소수

chapter 03

11.0010001111010111000010100011110101110000101000111 10...으로 계속 순환하는 순환소수가 됩니다.

조건 연산자와 증가 연산자

chapter 04

max는 21, a는 21, b는 30으로 출력됩니다. 조건 연산자는 선택된 식만을 계산하기 때문에 ++b는 계산되지 않습니다.

넓은 범위로의 캐스트 연산은 안전할까?

chapter 04

좁은 범위의 정수 타입이 표현하는 모든 수는 넓은 범위의 정수 타입으로도 표현할 수 있기 때문에 좁은 범위의 정수 타입을 넓은 범위의 정수 타입으로 변환할 때는 데이터 손실이 발생하지 않습니다. float 타입을 double 타입으로 변환할 때도 지수 비트수와 가수 비트수가 모두 늘어나기 때문에 데이터 손실이 발생하지 않습니다. 문제가 되는 것은 정수 타입을 부동소수점 타입으로 바꿀 경우뿐입니다.

float 타입과 double 타입은 정수 타입보다 표현 범위는 넓지만, 모든 비트들을 다 유효숫자 표현에 사용하는 것이 아니기 때문에 아주 큰 정수를 담을 때는 정확성에 손실이 생길 수도 있습니다. 예를 들어 int 타입은 MSB를 제외한 모든 비트(31 bits)를 유효숫자 표현에 사용하지만, float 타입은 가수 표현에 할당된 비트(23 bits)만 사용하기 때문에 아주 큰 int 타입 값(예를 들어 1234567890)을 float 타입으로 변환하면 유효숫자의 일부를 잃을 수 있습니다. 마찬가지 이유로 long 타입을 double 타입으로 변환할 때도 그런 일이 발생할 수 있습니다.

하지만 double 타입은 가수 표현에 할당된 비트 수(52 bits)가 int 타입의 유효숫자 표현 비트 수(31 bits)보다 많기 때문에 int 타입을 double 타입으로 변환할 때는 데이터 손실이 발생하지 않습니다. 마찬가지 이유로 byte, short, char 타입을 float, double 타입으로 변환할 때도 데이터 손실은 발생하지 않습니다.

private 필드의 장단점

chapter 05

stockNum 필드를 private 필드로 만들면 무분별한 필드 값 변경으로 인해 재고수량이 마이너스가 되는 상황을 막을 수 있습니다. 하지만 그렇게 하면 객체 외부에서 재고수량을 알 수 있는 방법이 없기 때문에, 이를 보완하기 위해서는 stockNum 필드의 값을 리턴하는 다음과 같은 메소드를 추가해야 합니다.

```
class GoodsStock {
    ....
    private int stockNum;
    ...
    int getStockNum() {
        return stockNum;
    }
}
```

객체를 만들지 못하게 하는 방법

chapter 05

클래스 안에 무조건 익셉션을 던지는 생성자를 선언해두면 그 클래스의 객체를 생성할 수 없게 됩니다.

```
class IntBytes {
    IntBytes() throws Exception {
        throw new Exception("객체를 생성할 수 없습니다.");
    }
    ...
}
```

또 다른 방법으로는 abstract 키워드를 사용하는 방법과 생성자에 private 키워드를 사용하는 방법이 있는데, abstract 키워드를 사용하는 방법은 6장에서 배울 것이고 private 키워드를 사용하는 방법은 8장에서 배울 것입니다.

추상 클래스의 생성자 chapter 06

생성자는 클래스의 객체를 만들 때도 사용되지만, 서브클래스에서 호출되기도 합니다. 그렇기 때문에 추상 클래스에도 생성자를 선언해 두어야할 경우가 종종 있습니다. 예를 들어 CheckingAccount(예제 6-7), CreditLineAccount(예제 6-8), BonusPointAccount(예제 6-11)의 생성자에서는 슈퍼클래스인 Account의 생성자를 호출하고 있기 때문에 [예제 6-15]에서 생성자를 없애버리면 안됩니다.

좁은 접근 범위로의 메소드 오버라이딩 chapter 08

클래스 변수나 인터페이스 변수의 다형성을 활용하는 부분에서 문제가 일어날 수 있습니다. 자바 컴파일러는 컴파일을 할 때 변수의 타입을 가지고 문법 체크를 하는데, 만약 그 변수가 가리키는 객체가 변수의 메소드보다 좁은 접근 범위 메소드로 오버라이딩을 하고 있다면 실행 도중에 에러가 발생할 것입니다.

String 객체의 내용이 변경 가능하다면? chapter 09

똑같은 내용의 문자열 리터럴들은 하나의 String 객체를 공유하기 때문에, 한 문자열 리터럴에 가한 수정이 같은 내용의 다른 문자열 리터럴 내용도 바꾸게 됩니다.

finalize 메소드를 사용했을 때의 장단점 chapter 11

finalize 메소드를 이용하여 시스템 자원을 해제하면 그 메소드를 따로 호출할 필요도 없고, 객체가 존재하는 동안 시스템 자원을 자유로이 사용할 수 있어서 좋습니다. 하지만 프로그램의 전체 실행 시간에 비해 시스템 자원을 사용하는 시간이 아주 짧을 때 이런 방법을 사용하면 시스템 자원의 해제가 불필요하게 지연될 수 있습니다. 그런 식의 지연이 많아지면 프로그램의 다른 부분이나 다른 프로그램에서 필요로 하는 시스템 자원이 부족하거나 고갈되는 문제를 초래할 수 있습니다.

ArrayList와 LinkedList 비교

chapter 13

ArrayList 클래스는 배열에 데이터를 저장하는데, 배열의 앞쪽에 데이터를 추가하거나 삭제하면 나머지 데이터들이 한 칸씩 이동해야 합니다. 그렇기 때문에 배열의 앞쪽을 큐의 입구나 출구로 사용하면 프로그램의 성능이 떨어질 수 있습니다.

로컬 이너 클래스의 선언 위치

chapter 16

인터페이스의 메소드는 본체를 가질 수 없기 때문입니다.

이름 없는 이너 클래스의 생성자

chapter 16

생성자 선언은 클래스 이름으로 시작해야 하기 때문입니다.

isReady 필드 값을 체크하지 않는다면?

chapter 18

CarcThread가 계산을 아주 빨리 끝냈을 경우에 문제가 될 수 있습니다. 그럴 경우에는 wait 메소드보다 notify 메소드가 먼저 호출될 수 있는데, 그렇게 되면 notify 메소드가 보내는 신호는 아무 곳에도 가지 않고 그대로 소멸되어 버리고 맙니다. 그리고 그 후에 호출된 wait 메소드는 오지 않는 신호를 기다리면서 무한정 대기하는 상태가 됩니다.

더 완벽한 서버 프로그램 만들기

chapter 20

서버 프로그램에 종료 명령이 입력되기를 기다리는 스레드를 추가하면 됩니다.

JDBC 프로그램의 흐름 이해

chapter 21

테이블 내용을 읽다가 익셉션이 발생하면 Connection 객체가 close되지 않은 채로 프로그램이 끝날 수 있습니다.