Declarations and Access Control

Nilesh Dungarwal

Java Refresher

- Class
- Object
- State(instance variables)
- Behavior(methods)

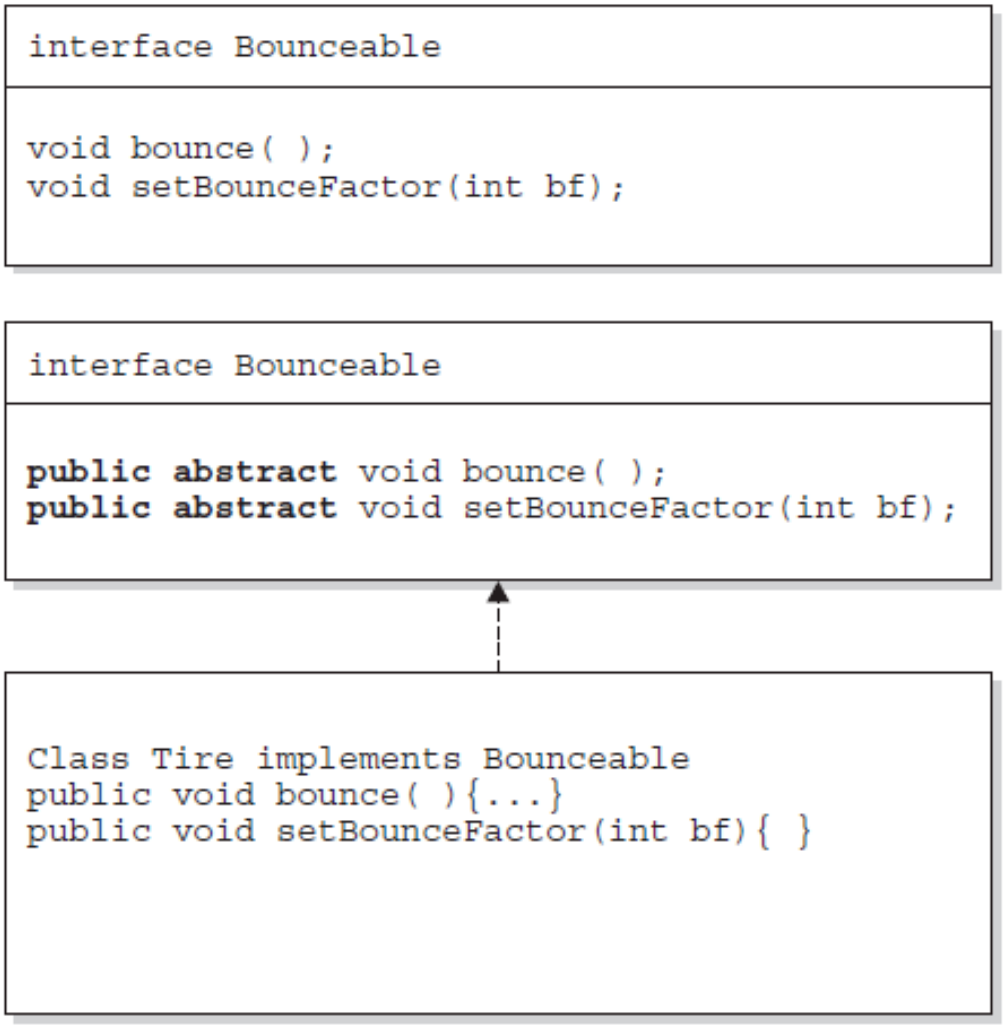
Java Refresher

- Identifiers and keywords
- Inheritance
- Finding other classes
- Always follow Java Code Conventions

Class Declarations and Modifiers

- Would this compile?
  - class MyClass{ }
- Class Access
  - public
  - default
  - protected
  - private
- Other Class Modifiers
  - strictfp
  - final
  - abstract

Declare Interfaces



Exam Watch

exam watch

Look for interface definitions that define constants, but without explicitly using the required modifiers. For example, the following are all identical:

```
public int x = 1;              // Looks non-static and non-final,
                               // but isn't!
int x = 1;                     // Looks default, non-final,
                               // non-static, but isn't!
static int x = 1;              // Doesn't show final or public
final int x = 1;               // Doesn't show static or public
public static int x = 1;           // Doesn't show final
public final int x = 1;            // Doesn't show static
static final int x = 1             // Doesn't show public
public static final int x = 1;     // what you get implicitly
```

Any combination of the required (but implicit) modifiers is legal, as is using no modifiers at all! On the exam, you can expect to see questions you won't be able to answer correctly unless you know, for example, that an interface variable is `final` *and can never be given a value by the implementing (or any other) class.*

Determining access to Class members

Visibility	Public	Protected	*Default*	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, *through inheritance*	No	No
From any non-subclass class outside the package	Yes	No	No	No

Method with Variable Argument Lists

Legal:

```
void doStuff(int... x) { }  // expects from 0 to many ints
                            // as parameters
void doStuff2(char c, int... x)  { }  // expects first a char,
                                      // then 0 to many ints
void doStuff3(Animal... animal) { }   // 0 to many Animals
```

Illegal:

```
void doStuff4(int x...) { }              // bad syntax
void doStuff5(int... x, char... y) { }  // too many var-args
void doStuff6(String... s, byte b) { }  // var-arg must be last
```

Variable declarations

- Primitives: char, boolean, byte, short, int, long
  double, float

- Reference variables: used to refer to (or
  access) to an object

Range of numberic primitives

Type	Bits	Bytes	Minimum Range	Maximum Range
byte	8	1	-2^7	2^7-1
short	16	2	-2^{15}	$2^{15}-1$
int	32	4	-2^{31}	$2^{31}-1$
long	64	8	-2^{63}	$2^{63}-1$
float	32	4	n/a	n/a
double	64	8	n/a	n/a

Array Declarations

- Arrays are objects that store multiple variables of same types

Declaring an Array of Primitives

```
int[] key;  // Square brackets before name (recommended)
int key []; // Square brackets after name (legal but less
            // readable)
```

Declaring an Array of Object References

```
Thread[] threads;  // Recommended
Thread threads []; // Legal but less readable
```

Exam Watch

exam Watch

It is never legal to include the size of the array in your declaration. Yes, we know you can do that in some other languages, which is why you might see a question or two that include code similar to the following:

```
int[5] scores;
```

The preceding code won't compile. Remember, the JVM doesn't allocate space until you actually instantiate the array object. That's when size matters.

Other variable types

- Transient variables
- Volatile variables
- Static variables and methods

Declare Enums

```
enum CoffeeSize { BIG, HUGE, OVERWHELMING }  // this cannot be
                                             // private or protected

class Coffee {
   CoffeeSize size;
}

public class CoffeeTest1 {
   public static void main(String[] args) {
      Coffee drink = new Coffee();
      drink.size = CoffeeSize.BIG;        // enum outside class
   }
}
```

Enums

The following is NOT legal:

```
public class CoffeeTest1 {
  public static void main(String[] args) {
    enum CoffeeSize { BIG, HUGE, OVERWHELMING } // WRONG! Cannot
                                                // declare enums
                                                // in methods

    Coffee drink = new Coffee();
    drink.size = CoffeeSize.BIG;
  }
}
```

Question

Given:

```
1. class Voop {
2.   public static void main(String [] args) {
3.     doStuff(1);
4.     doStuff(1,2);
5.   }
6.   // insert code here
7. }
```

Which, inserted independently at line 6, will compile? (Choose all that apply.)

A. `static void doStuff(int... doArgs) { }`

B. `static void doStuff(int[] doArgs) { }`

C. `static void doStuff(int doArgs...) { }`

D. `static void doStuff(int... doArgs, int y) { }`

E. `static void doStuff(int x, int... doArgs) { }`

Answer

Answer:

☑ **A** and **E** use valid var-args syntax.

☒ **B** and **C** are invalid var-arg syntax, and **D** is invalid because the var-arg must be the last of a method's arguments. (Objective 1.4)

Question

Given:

```
4. class Announce {
5.   public static void main(String[] args) {
6.     for(int __x = 0; __x < 3; __x++) ;
7.     int #1b = 7;
8.     long [] x [5];
9.     Boolean []ba[];
10.    enum Traffic { RED, YELLOW, GREEN };
11.  }
12. }
```

What is the result? (Choose all that apply.)

A. Compilation succeeds

B. Compilation fails with an error on line 6

C. Compilation fails with an error on line 7

D. Compilation fails with an error on line 8

E. Compilation fails with an error on line 9

F. Compilation fails with an error on line 10

Answer

Answer:

☑ **C, D,** and **F** are correct. Variable names cannot begin with a #, an array declaration can't include a size without an instantiation, and enums can't be declared within a method.

☒ **A, B,** and **E** are incorrect based on the above information. (Objective 1.3)

Contact Info

- trainers@finaldesk.com
- rishabh@finaldesk.com
- nilesh@finaldesk.com
- jignesh@finaldesk.com
- yash@finaldesk.com
- anand@finaldesk.com