

The (*most correct*) answers are in blue.
Comments are in red.
Class names (for actual code) are in green.

The following program is compiled then run with this command line:

```
java Demo alpha beta gamma

public class Demo {
    public static void main( String [] args ) {
        final int n = 3 ;
        System.out.println( "The word is " + args[ n ] ) ;
    }
}
```

What happens?

- "The word is beta" is written to standard output.
- "The word is gamma" is written to standard output.
- The runtime system reports an `ArrayIndexOutOfBoundsException` in the main method.**
args.length is 3
args[0] is "alpha"
args[1] is "beta"
args[2] is "gamma"
- The runtime system reports an `NullPointerException` in the main method.

Demo

Is it legal to access a static method using an instance of the class?

- No
- Yes**

CallStatic

Is this legal?

```
public static void main( String [] fred )
```

- No
- Yes**

You can call a variable whatever you like (as long as it is a legal Java variable name, that is.)

Fred

Can one object access a `private` variable of another object of the same class?

No Yes

Secret

Can a method have more than one *access modifier* (`public`, `protected`, `private`)?

No Yes

OnlyOne

Can a subclass access the `private` member variables of its own superclass?

No Yes

SuperPrivate

Is `cast` a keyword?

No Yes

This used to be a reserved word, but not anymore (I use version 1.6 of the JDK).

A cast is a conversion from a generic class type which represents a superclass of a more specific class type.

Cast

All Java keywords are written in lower case.

False True

Is this legal?

```
int x [ ] = { 2, 7, 4, 9 } ;
```

No Yes

X4

When an object is created using `new`, its constructor will not complete until all constructors higher on the inheritance tree have completed.

False True

Up

What is the output for the following lines of code?

```
1: System.out.println( "" + 2 + 3 ) ;  
   ( "" + Integer.toString( 2 ) ) + Integer.toString( 3 )  
2: System.out.println( 2 + 3 ) ;  
   Integer.toString( 2 + 3 )  
3: System.out.println( 2 + 3 + "" ) ;  
   Integer.toString( 2 + 3 ) + ""  
4: System.out.println( 2 + "" +3 ) ;  
   ( Integer.toString( 2 ) + "" ) + Integer.toString( 3 )
```

- Compilation error at line 3
- Prints 23, 5, 5 and 23.
- Prints 5, 5, 5 and 23.
- Prints 23, 5, 23 and 23.

Out23

Given the following class definitions:

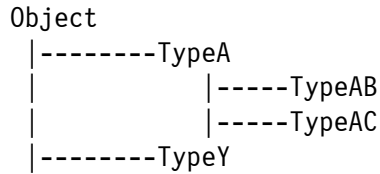
```
class BaseWidget extends Object  
{  
    String name = "BaseWidget" ;  
    void speak( ) { System.out.println( "I am a " + name ) ; }  
}  
class TypeAWidget extends BaseWidget  
{  
    TypeAWidget( ) { name = "TypeA" ; }  
}
```

What will happen when we try to compile and run the following method?

- ```
1: public void WhoAreYou() {
2: Object A = new TypeAWidget() ;
3: ((BaseWidget) A).speak() ;
4: }
```
- The method will compile and output "I am a BaseWidget".
  - The method will compile and output "I am a TypeA".
  - The compiler will object to line 3.
  - A runtime cast exception will be generated in line 2.
  - A runtime cast exception will be generated in line 3.

### TypeA

Given the following hierarchical relationship of several classes:



(a) And given the following method definition:

```
public void sayType(Object x)
{
 // Missing right braces at the end of each line!
 if (x instanceof Object) { System.out.print("Object,") ; }
 if (x instanceof TypeA) { System.out.print("TypeA,") ; }
 if (x instanceof TypeAB) { System.out.print("TypeAB,") ; }
 if (x instanceof TypeAC) { System.out.print("TypeAC,") ; }
}
```

What would the program output be if the following line was executed:

```
sayType(new TypeAB()) ;
```

- Object,
- Object,TypeA,TypeAB, *Missing comma*
- TypeAB,

(b) And given that the following code exists:

in TypeA.java:

```
public void sayType() { System.out.print("TypeA,") ; }
```

in TypeAB.java:

```
public void sayType() { System.out.print("TypeAB,") ; }
```

in TypeAC.java:

```
public void sayType() { System.out.print("TypeAC,") ; }
```

What is the output of the following code?

```
1: TypeA [] objects = {
2: new TypeA(), new TypeAB(), new TypeAC()
3: } ;
4: for (int i = 0 ; i < objects.length ; i++) {
5: objects[i].sayType() ;
6: }
```

- TypeA,TypeAB,TypeAC,
- TypeA,TypeA,TypeA,
- TypeA,Class cast exception at line 5

Types

A method to compute the sum of all elements in an array of `int` is needed. The following proposed method is incomplete -- select the correct statement for line 3 from the options provided.

```
1: public int total(int [] x) {
2: int i, t = 0 ;
3: // select statement to go here
4: { t += x[i++] ; }
5: return t ;
6: }
```

- `for ( int i = 0 ; i < x.length ; )` **compile error: i was already declared**
- `for ( i = 0 ; i < x.length ; )` **correct total**
- `for ( i = 0 ; i < x.length ; i++ )` **incorrect total**
- `for ( i = 1 ; i <= x.length ; i++ )` **incorrect total**

### Total

Trying to compile the following source code produces a compiler error message to the effect that the variable `tmp` may not have been initialized.

```
1: class Value {
2: public static final String msg = "Value is " ;
3: public void showValue(int n) {
4: String tmp ;
5: if (n > 0) { tmp = "positive" ; }
6: System.out.println(msg + tmp) ;
7: }
8: }
```

Which of the following changes would eliminate this error? Select all that apply.

- Make line 4 read  
4: `String tmp = null ;` **works, but poor output**
- Make line 4 read  
4: `String tmp = "" ;` **works, but poor output**
- Insert a line following 5**  
6: `else { tmp = "not positive" ; }`  
**This is the best choice, as it also corrects the output of the program for all values of n.**
- Remove line 4 and insert a new line after 2 so that `tmp` becomes a member variable instead of a local variable in `showValue`.  
3: `String tmp ;` **works, but poor output**

### Values

Given the class definition shown:

```
public class DerivedDemo extends Demo
{
 int M, N, L ;

 public DerivedDemo(int x, int y) {
 M = x ; N = y ;
 }

 public DerivedDemo(int x) {
 super(x) ;
 }
}
```

Which of the following constructor signatures **MUST** exist in the Demo class for DerivedDemo to compile correctly?

- public Demo( int a, int b )
- public Demo( int c )** explicitly called by the second constructor
- public Demo( )** implicitly called by the first constructor
- There is no required constructor in Demo.

### DerivedDemo

The following lists the complete contents of the file named Derived.java.

```
1: public class Base extends Object {
2: String objType ;
3: public Base() { objType = "I am a Base type" ; }
4: }
5:
6: public class Derived extends Base {
7: public Derived() { objType = "I am a Derived type" ; }
8:
9: public static void main(String args []) {
10: Derived D = new Derived() ;
11: }
12: }
```

What will happen when this file is compiled?

- Two class files, Base.class and Derived.class will be created.
- The compiler will object to line one.** Only one public class per .java file.
- The compiler will object to line six.

### Derived

What will happen if you compile/run the following code?

```
1: public class OverMain
2: {
3: static final String str1 = "main method with String[] args";
4: static final String str2 = "main method with int[] args";
5:
6: public static void main(String [] args)
7: {
8: System.out.println(str1);
9: }
10:
11: public static void main(int [] args)
12: {
13: System.out.println(str2);
14: }
15: }
```

- Duplicate method main(), compilation error at line 6.
- Duplicate method main(), compilation error at line 11.
- Prints "main method with String[] args". The JVM *always* runs this method.
- Prints "main method with int[] args".

**OverMain**

What is output by the following code?

```
class Exam
{
 protected String difficultyLevel = "easy" ;

 public void printDifficultyLevel()
 {
 System.out.println(difficultyLevel) ;
 }
}

public class BoldinExam extends Exam
{
 private String difficultyLevel = "killing" ;

 public static void main(String [] args)
 {
 BoldinExam myExam = new BoldinExam() ;
 myExam.printDifficultyLevel() ;
 }
}
```

**easy**

BoldinExam's difficultyLevel hides Exam's difficulty level, but only for new code in BoldinExam. Exam's difficultyLevel, which is simply inherited (i.e., not copied as new code in BoldinExam by the compiler), uses the difficulty level in Exam.

If I add the line:

```
System.out.println(myExam.difficultyLevel) ;
```

to main(), it prints out "killing". ;-)

**killing**

**BoldinExam**