1.

What will be the output of the program?

```
public class Test
{
    public static void main(String[] args)
    {
        int x = 0;
        assert (x > 0) ? "assertion failed" : "assertion passed" ;
        System.out.println("finished");
    }
}
```

A. finished

B. Compiliation fails.

C. An AssertionError is thrown and finished is output.

D. An AssertionError is thrown with the message "assertion failed."

Answer & Explanation

Answer: Option B

Explanation:

Compilation Fails. You can't use the Assert statement in a similar way to the ternary operator. Don't confuse.

View Answer Workspace Report Discuss in Forum

2.

```
public class Test
```

```
{
   public void foo()
   {
      assert false; /* Line 5 */
      assert false; /* Line 6 */
   }
   public void bar()
   {
      while(true)
      {
         assert false; /* Line 12 */
      }
      assert false;  /* Line 14 */
   }
}
```

What causes compilation to fail?

A. Line 5

B. Line 6

C. Line 12

D. Line 14

Answer & Explanation

Answer: Option D

Explanation:

Option D is correct. Compilation fails because of an unreachable statement at line 14. It is a compile-time error if a statement cannot be executed because it is unreachable. The question is now, why is line 20 unreachable? If it is because of the assert then surely line 6 would also be unreachable. The answer must be something other than assert.

Examine the following:

A while statement can complete normally if and only if at least one of the following is true:

- The while statement is reachable and the condition expression is not a constant expression with value true.

-There is a reachable break statement that exits the while statement.

The while statement at line 11 is infinite and there is no break statement therefore line 14 is unreachable. You can test this with the following code:

```
public class Test80
{
   public void foo()
   {
      assert false;
      assert false;
   }
   public void bar()
   {
      while(true)
      {
         assert false;
         break;
      }
      assert false;
   }
}
```

View Answer Workspace Report Discuss in Forum

3.

What will be the output of the program?

```
public class Test
{
   public static int y;
   public static void foo(int x)
   {
      System.out.print("foo ");
      y = x;
   }
   public static int bar(int z)
   {
      System.out.print("bar ");
      return y = z;
   }
   public static void main(String [] args )
   {
      int t = 0;
      assert t > 0 : bar(7);
      assert t > 1 : foo(8); /* Line 18 */
      System.out.println("done ");
   }
}
```

A. bar

B. bar done

C. foo done

D. Compilation fails

Answer & Explanation

Answer: Option D

Explanation:

The foo() method returns void. It is a perfectly acceptable method, but because it returns void it cannot be used in an assert statement, so line 18 will not compile.

View Answer Workspace Report Discuss in Forum

4.

What will be the output of the program (when you run with the -ea option) ?

```
public class Test
{
  public static void main(String[] args)
  {
    int x = 0;
    assert (x > 0) : "assertion failed"; /* Line 6 */
    System.out.println("finished");
  }
}
```

A. finished

B. Compilation fails.

C. An AssertionError is thrown.

D. An AssertionError is thrown and finished is output.

Answer & Explanation

Answer: Option C

Explanation:

An assertion Error is thrown as normal giving the output "assertion failed". The word "finished" is not printed (ensure you run with the -ea option)

Assertion failures are generally labeled in the stack trace with the file and line number from which they were thrown, and also in this case with the error's detail message "assertion failed". The detail message is supplied by the assert statement in line 6.

View Answer Workspace Report Discuss in Forum

5.

```
public class Test2
{
    public static int x;
    public static int foo(int y)
    {
        return y * 2;
    }
    public static void main(String [] args)
    {
        int z = 5;
        assert z > 0; /* Line 11 */
        assert z > 2: foo(z); /* Line 12 */
        if ( z < 7 )
            assert z > 4; /* Line 14 */

        switch (z)
        {
            case 4: System.out.println("4 ");
            case 5: System.out.println("5 ");
            default: assert z < 10;
        }

        if ( z < 10 )
            assert z > 4: z++; /* Line 22 */
        System.out.println(z);
    }
}
```

which line is an example of an inappropriate use of assertions?

A. Line 11

B. Line 12

C. Line 14

D. Line 22

Answer & Explanation

Answer: Option D

Explanation:

Assert statements should not cause side effects. Line 22 changes the value of z if the assert statement is false.

Option A is fine; a second expression in an assert statement is not required.

Option B is fine because it is perfectly acceptable to call a method with the second expression of an assert statement.

Option C is fine because it is proper to call an assert statement conditionally.

1.

Which of the following statements is true?

A. If assertions are compiled into a source file, and if no flags are included at runtime, assertions will execute by default.

B. As of Java version 1.4, assertion statements are compiled by default.

C. With the proper use of runtime arguments, it is possible to instruct the VM to disable assertions for a certain class, and to enable assertions for a certain package, at the same time.

D. When evaluating command-line arguments, the VM gives -ea flags precedence over -da flags.

Answer & Explanation

Answer: Option C

Explanation:

Option C is true because multiple VM flags can be used on a single invocation of a Java program.

Option A is incorrect because at runtime assertions are ignored by default.

Option B is incorrect because as of Java 1.4 you must add the argument -source 1.4 to the command line if you want the compiler to compile assertion statements.

Option D is incorrect because the VM evaluates all assertion flags left to right.

View Answer Workspace Report Discuss in Forum

2.

Which statement is true?

A. Assertions can be enabled or disabled on a class-by-class basis.

B. Conditional compilation is used to allow tested classes to run at full speed.

C. Assertions are appropriate for checking the validity of arguments in a method.

D. The programmer can choose to execute a return statement or to throw an exception if an assertion fails.

Answer & Explanation

Answer: Option A

Explanation:

Option A is correct. The assertion status can be set for a named top-level class and any nested classes contained therein. This setting takes precedence over the class loader's default assertion status, and over any applicable per-package default. If the named class is not a top-level class, the change of status will have no effect on the actual assertion status of any class.

Option B is wrong. Is there such a thing as conditional compilation in Java?

Option C is wrong. For private methods - yes. But do not use assertions to check the parameters of a public method. An assert is inappropriate in public methods because the method guarantees that it will always enforce the argument checks. A public method must check its arguments whether or not assertions are enabled. Further, the assert construct does not throw an exception of the specified type. It can throw only an AssertionError.

Option D is wrong. Because you're never supposed to handle an assertion failure. That means don't catch it with a catch clause and attempt to recover.

View Answer Workspace Report Discuss in Forum

3.

Which statement is true about assertions in the Java programming language?

A. Assertion expressions should not contain side effects.

B. Assertion expression values can be any primitive type.

C. Assertions should be used for enforcing preconditions on public methods.

D. An AssertionError thrown as a result of a failed assertion should always be handled by the enclosing method.

Answer & Explanation

Answer: Option A

Explanation:

Option A is correct. Because assertions may be disabled, programs must not assume that the boolean expressions contained in assertions will be evaluated. Thus these expressions should be free of side effects. That is, evaluating such an expression should not affect any state that is visible after the evaluation is complete. Although it is not illegal for a boolean expression contained in an assertion to have a side effect, it is generally inappropriate, as it could cause program behaviour to vary depending on whether assertions are enabled or disabled.

Assertion checking may be disabled for increased performance. Typically, assertion checking is enabled during program development and testing and disabled for deployment.

Option B is wrong. Because you assert that something is "true". True is Boolean. So, an expression must evaluate to Boolean, not int or byte or anything else. Use the same rules for an assertion expression that you would use for a while condition.

Option C is wrong. Usually, enforcing a precondition on a public method is done by condition-checking code that you write yourself, to give you specific exceptions.

Option D is wrong. "You're never supposed to handle an assertion failure"

Not all legal uses of assertions are considered appropriate. As with so much of Java, you can abuse the intended use for assertions, despite the best efforts of Sun's Java engineers to discourage you. For example, you're never supposed to handle an assertion failure. That means don't catch it with a catch clause and attempt to recover. Legally, however, AssertionError is a subclass of Throwable, so it can be caught. But just don't do it! If you're going to try to recover from something, it should be an exception. To discourage you from trying to substitute an assertion for an exception, the AssertionError doesn't provide access to the object that generated it. All you get is the String message.

View Answer Workspace Report Discuss in Forum

4.

Which of the following statements is true?

A. It is sometimes good practice to throw an AssertionError explicitly.

B. Private getter() and setter() methods should not use assertions to verify arguments.

C. If an AssertionError is thrown in a try-catch block, the finally block will be bypassed.

D. It is proper to handle assertion statement failures using a catch (AssertionException ae) block.

Answer & Explanation

Answer: Option A

Explanation:

Option A is correct because it is sometimes advisable to thrown an assertion error even if assertions have been disabled.

Option B is incorrect because it is considered appropriate to check argument values in private methods using assertions.

Option C is incorrect; finally is never bypassed.

Option D is incorrect because AssertionErrors should never be handled.

View Answer Workspace Report Discuss in Forum

5.

Which of the following statements is true?

A. In an assert statement, the expression after the colon (:) can be any Java expression.

B. If a switch block has no default, adding an assert default is considered appropriate.

C. In an assert statement, if the expression after the colon (:) does not have a value, the assert's error message will be empty.

D. It is appropriate to handle assertion failures using a catch clause.

Answer & Explanation

Answer: Option B

Explanation:

Adding an assertion statement to a switch statement that previously had no default case is considered an excellent use of the assert mechanism.

Option A is incorrect because only Java expressions that return a value can be used. For instance, a method that returns void is illegal.

Option C is incorrect because the expression after the colon must have a value.

Option D is incorrect because assertions throw errors and not exceptions, and assertion errors do cause program termination and should not be handled.

6.

Which three statements are true?

Assertion checking is typically enabled when a program is deployed.

It is never appropriate to write code to handle failure of an assert statement.

Assertion checking is typically enabled during program development and testing.

Assertion checking can be selectively enabled or disabled on a per-package basis, but not on a per-class basis.

Assertion checking can be selectively enabled or disabled on both a per-package basis and a per-class basis.

A. 1, 2 and 4

B. 2, 3 and 5

C. 3, 4 and 5

D. 1, 2 and 5

Answer & Explanation

Answer: Option B

Explanation:

(1) is wrong. It's just not true.

(2) is correct. You're never supposed to handle an assertion failure.

(3) is correct. Assertions let you test your assumptions during development, but the assertion code—in effect—evaporates when the program is deployed, leaving behind no overhead or debugging code to track down and remove.

(4) is wrong. See the explanation for (5) below.

(5) is correct. Assertion checking can be selectively enabled or disabled on a per-package basis. Note that the package default assertion status determines the assertion status for classes initialized in the future that belong to the named package or any of its "subpackages".

The assertion status can be set for a named top-level class and any nested classes contained therein. This setting takes precedence over the class loader's default assertion status, and over any applicable per-package default. If the named class is not a top-level class, the change of status will have no effect on the actual assertion status of any class.

1.

```
public class Test2

{

   public static int x;

   public static int foo(int y)

   {

      return y * 2;

   }

   public static void main(String [] args)

   {

      int z = 5;
```

```
        assert z > 0; /* Line 11 */
        assert z > 2: foo(z); /* Line 12 */
        if ( z < 7 )
            assert z > 4; /* Line 14 */

        switch (z)
        {
            case 4: System.out.println("4 ");
            case 5: System.out.println("5 ");
            default: assert z < 10;
        }

        if ( z < 10 )
            assert z > 4: z++; /* Line 22 */
        System.out.println(z);
    }
}
```

which line is an example of an inappropriate use of assertions?

A. Line 11

B. Line 12

C. Line 14

D. Line 22

Answer & Explanation

Answer: Option D

Explanation:

Assert statements should not cause side effects. Line 22 changes the value of z if the assert statement is false.

Option A is fine; a second expression in an assert statement is not required.

Option B is fine because it is perfectly acceptable to call a method with the second expression of an assert statement.

Option C is fine because it is proper to call an assert statement conditionally.

View Answer Workspace Report Discuss in Forum

2.

```
public class Test
{
    public void foo()
    {
        assert false; /* Line 5 */
        assert false; /* Line 6 */
    }
    public void bar()
    {
        while(true)
        {
            assert false; /* Line 12 */
        }
        assert false;  /* Line 14 */
    }
}
```

What causes compilation to fail?

A. Line 5

B. Line 6

C. Line 12

D. Line 14

Answer & Explanation

Answer: Option D

Explanation:

Option D is correct. Compilation fails because of an unreachable statement at line 14. It is a compile-time error if a statement cannot be executed because it is unreachable. The question is now, why is line 20 unreachable? If it is because of the assert then surely line 6 would also be unreachable. The answer must be something other than assert.

Examine the following:

A while statement can complete normally if and only if at least one of the following is true:

- The while statement is reachable and the condition expression is not a constant expression with value true.

-There is a reachable break statement that exits the while statement.

The while statement at line 11 is infinite and there is no break statement therefore line 14 is unreachable. You can test this with the following code:

```
public class Test80
{
  public void foo()
  {
    assert false;
    assert false;
    }
    public void bar()
    {
        while(true)
        {
            assert false;
            break;
        }
        assert false;
    }
}
```