01

Q: 02 Given a method that must ensure that its parameter is not null:

11. public void someMethod(Object value) {

12. // check for null value

...

20. System.out.println(value.getClass());

21. }

What, inserted at line 12, is the appropriate way to handle a null value?

A. assert value == null;

B. assert value != null, "value is null";

C. if (value == null) {

throw new AssertionException("value is null");

}

D. if (value == null) {

throw new IllegalArgumentException("value is null");

}

Answer: D

02

05. Given two files:

1. class One {

2. public static void main(String[] args) {

3. int assert = 0;

4. }

5. }

1. class Two {

2. public static void main(String[] args) {

3. assert(false);

4. }

5. }

And the four command-line invocations:

javac -source 1.3 One.java

javac -source 1.4 One.java

javac -source 1.3 Two.java

javac -source 1.4 Two.java

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

A. Only one compilation will succeed.

B. Exactly two compilations will succeed.

C. Exactly three compilations will succeed.

D. All four compilations will succeed.

E. No compiler warnings will be produced.

F. At least one compiler warning will be produced.

Answer:

-> B and F are correct. Class One will compile (and issue a warning) using the 1.3 flag, andclass Two will compile using the 1.4 flag.

-> A, C, D, and E are incorrect based on the above. (Objective 2.3)

03

11. static void test() throws Error {  
12. if (true) throw new AssertionError();  
13. System.out.print("test ");  
14. }  
15. public static void main(String[] args) {  
16. try { test(); }  
17. catch (Exception ex) { System.out.print("exception "); }  
18. System.out.print("end ");  
19. }

What is the result?

A. end  
B. Compilation fails.  
C. exception end  
D. exception test end  
E. A Throwable is thrown by main.   
F. An Exception is thrown by main.

Answer: E (Error estende de Throwable e não está sendo capturado pelo catch)  
  
04

11. Float pi = new Float(3.14f);  
12. if (pi > 3) {  
13. System.out.print("pi is bigger than 3. ");  
14. }  
15. else {  
16. System.out.print("pi is not bigger than 3. ");  
17. }  
18. finally {  
19. System.out.println("Have a nice day.");  
20. }

What is the result?

A. Compilation fails.  
B. pi is bigger than 3.  
C. An exception occurs at runtime.  
D. pi is bigger than 3. Have a nice day.  
E. pi is not bigger than 3. Have a nice day.

Answer: A (um finally sempre está atrelado a um catch)  
  
05

11. public static void main(String[] args) {  
12. try {  
13. args = null;  
14. args[0] = "test";  
15. System.out.println(args[0]);  
16. } catch (Exception ex) {  
17. System.out.println("Exception");  
18. } catch (NullPointerException npe) {  
19. System.out.println("NullPointerException");  
20. }  
21. }

What is the result?

A. test  
B. Exception  
C. Compilation fails.  
D. NullPointerException

Answer: C (Uma exceção mais abrangente está sendo capturada antes de uma menos abrangente)  
  
  
06

33. try {  
34. // some code here  
35. } catch (NullPointerException e1) {  
36. System.out.print("a");  
37. } catch (RuntimeException e2) {  
38. System.out.print("b");  
39. } finally {  
40. System.out.print("c");  
41. }

What is the result if a NullPointerException occurs on line 34?

A. c  
B. a  
C. ab  
D. ac  
E. bc  
F. abc

Answer: D (primeiro catch e em seguida o finally)  
  
07

10. public class Foo {  
11. static int[] a;  
12. static { a[0]=2; }  
13. public static void main( String[] args ) {}  
14. }

Which exception or error will be thrown when a programmer attempts to run this code?

A. java.lang.StackOverflowError  
B. java.lang.IllegalStateException  
C. java.lang.ExceptionInInitializerError  
D. java.lang.ArrayIndexOutOfBoundsException

Answer: C (Erro dentro do bloco estático. Acessando uma variável nula.)

08

11.classA *{*  
12. public void process() *{* System.out.print(”A “); *} }*  
13. class B extends A *{*  
14. public void process() throws RuntimeException *{*  
15. super.process();  
16. if (true) throw new RuntimeException();  
17. System.out.print(“B”); *}}*  
18. public static void main(String[] args) *{*  
19. try *{* ((A)new B()).process(); *}*  
20. catch (Exception e) *{* System.out.print(”Exception “); *}*  
21. *}*

What is the result?

A. Exception  
B. A Exception  
C. A Exception B  
D. A B Exception  
E. Compilation fails because of an error in line 14.  
F. Compilation fails because of an error in line 19.

Answer: B (Imprime o A na 15 e levanta a exceção que é tratada na linha 20)

09

class Plane {  
static String s = "-";  
public static void main(String[] args) {  
new Plane().s1();  
System.out.println(s);  
}  
void s1() {  
try { s2(); }  
catch (Exception e) { s += "c"; }  
}  
void s2() throws Exception {  
s3(); s += "2";  
s3(); s += "2b";  
}  
void s3() throws Exception {  
throw new Exception();  
} }

What is the result?

A. -  
B. -c  
C. -c2  
D. -2c  
E. -c22b  
F. -2c2b  
G. -2c2bc  
H. Compilation fails.

Answer:  
-> B is correct. Once s3() throws the exception to s2(), s2() throws it to s1(), and no  
more of s2()’s code will be executed.  
-> A, C, D, E, F, G, and H are incorrect based on the above.  
  
10

try { int x = Integer.parseInt("two"); }

Which could be used to create an appropriate catch block? (Choose all that apply.)

A. ClassCastException  
B. IllegalStateException  
C. NumberFormatException  
D. IllegalArgumentException  
E. ExceptionInInitializerError  
F. ArrayIndexOutOfBoundsException

Answer:  
-> C and D are correct. Integer.parseInt can throw a NumberFormatException, and IllegalArgumentException is its superclass (i.e., a broader exception).  
-> A, B, E, and F are not in NumberFormatException’s class hierarchy.  
  
11

**class** Emu {

**static** String *s* = "-";

**public** **static** **void** main(String[] args) {

**try** {

**throw** **new** Exception();

} **catch** (Exception e) {

**try** {

**try** {

**throw** **new** Exception();

} **catch** (Exception ex) {

*s* += "ic ";

}

**throw** **new** Exception();

} **catch** (Exception x) {

*s* += "mc ";

} **finally** {

*s* += "mf ";

}

} **finally** {

*s* += "of ";

}

System.*out*.println(*s*);

}

}

What is the result?

A. -ic of  
B. -mf of  
C. -mc mf  
D. -ic mf of  
E. -ic mc mf of  
F. -ic mc of mf  
G. Compilation fails.

Answer:  
-> E is correct. There is no problem nesting try / catch blocks. As is normal, when an exception is thrown, the code in the catch block runs, then the code in the finally block runs.  
-> A, B, C, D, and F are incorrect based on the above.

12

Which two are true? (Choose two.)

A. A finalizer may NOT be invoked explicitly.  
B. The finalize method declared in class Object takes no action.  
C. super.finalize() is called implicitly by any overriding finalize method.  
D. The finalize method for a given object will be called no more than  
once by the garbage collector.  
E. The order in which finalize will be called on two objects is based on  
the order in which the two objects became finalizable.

Answer: BD

13

Which is true? (Choose all that apply.)

A. The invocation of an object’s finalize() method is always the last thing that happens before an object is garbage collected (GCed).  
B. When a stack variable goes out of scope it is eligible for GC.  
C. Some reference variables live on the stack, and some live on the heap.  
D. Only objects that have no reference variables referring to them can be eligible for GC.  
E. It’s possible to request the GC via methods in either java.lang.Runtime or  
java.lang.System classes.

Answer:  
-> C and E are correct. When an object has a reference variable, the reference variable lives inside the object, on the heap.  
->A is incorrect, because if, the first time an object’s finalize() method runs, the object is saved from the GC, then the second time that object is about to be GCed, finalize() will not run. B is incorrect—stack variables are not dealt with by the GC. D is incorrect because objects can live in "islands of isolation" and be GC eligible.  
  
14

1. class Eco {  
2. public static void main(String[] args) {  
3. Eco e1 = new Eco();  
4. Eco e2 = new Eco();  
5. Eco e3 = new Eco();  
6. e3.e = e2;  
7. e1.e = e3;  
8. e2 = null;  
9. e3 = null;  
10. e2.e = e1;  
11. e1 = null;  
12. }  
13. Eco e;  
14. }

At what point is only a single object eligible for GC?

A. After line 8 runs.  
B. After line 9 runs.  
C. After line 10 runs.  
D. After line 11 runs.  
E. Compilation fails.  
F. Never in this program.  
G. An exception is thrown at runtime.

Answer:  
-> G is correct. An error at line 10 causes a NullPointerException to be thrown because e2 was set to null in line 8. If line 10 was moved between lines 7 and 8, then F would be correct, because until the last reference is nulled none of the objects is eligible, and once the last reference is nulled, all three are eligible.  
-> A, B, C, D, E, and F are incorrect based on the above.

15

class Dec26 {

public static void main(String[] args) {

short a1 = 6;

new Dec26().go(a1);

new Dec26().go(new Integer(7));

}

void go(Short x) { System.*out*.print("S "); }

void go(Long x) { System.*out*.print("L "); }

void go(int x) { System.*out*.print("i "); }

void go(Number n) { System.*out*.print("N "); }

}

a) S N

b) S i

c) i N

letra C) i N

16

class Boxing3{

static void m(Integer i){System.*out*.println("Integer");}

static void m(Integer... i){System.*out*.println("Integer...");}

public static void main(String args[]){

int a=30;

*m*(a);

}

}

a) Integer

b) Integer...

Resposta: A

17

Given:

10. public class Foo implements java.io.Serializable {

11. private int x;

12. public int getX() { return x; }

12.publicFoo(int x){this.x=x; }

13. private void writeObject( ObjectOutputStream s)

14. throws IOException {

15. // insert code here

16. }

17. }

Which code fragment, inserted at line 15, will allow Foo objects to be

correctly serialized and deserialized?

A. s.writeInt(x);

B. s.serialize(x);

C. s.writeObject(x);

D. s.defaultWriteObject();

Answer: D

18

13. Given:

import java.io.\*;

class Player {

Player() { System.out.print("p"); }

}

class CardPlayer extends Player implements Serializable {

CardPlayer() {

System.out.print("c");

}

public static void main(String[] args) {

CardPlayer c1 = new CardPlayer();

try {

FileOutputStream fos = new FileOutputStream("play.txt");

ObjectOutputStream os = new ObjectOutputStream(fos);

os.writeObject(c1);

os.close();

FileInputStream fis = new FileInputStream("play.txt");

ObjectInputStream is = new ObjectInputStream(fis);

CardPlayer c2 = (CardPlayer) is.readObject();

is.close();

} catch (Exception x ) {

}

}

}

What is the result?

A. pc B. pcc

C. pcp D. pcpc

E. Compilation fails. F. An exception is thrown at runtime.

Answer:

-> C is correct. It's okay for a class to implement Serializable even if its superclass doesn't. However, when you deserialize such an object, the non-serializable superclass must run its constructor. Remember, constructors don't run on deserialized classes that implement Serializable.

-> A, B, D, E, and F are incorrect based on the above.

19

15. Given:

import java.io.\*;

class Keyboard { }

public class Computer implements Serializable {

private Keyboard k = new Keyboard();

public static void main(String[] args) {

Computer c = new Computer();

c.storeIt(c);

}

void storeIt(Computer c) {

try {

ObjectOutputStream os = new ObjectOutputStream(

new FileOutputStream("myFile"));

os.writeObject(c);

os.close();

System.out.println("done");

} catch (Exception x) {System.out.println("exc"); }

}

}

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

A. exc

B. done

C. Compilation fails.

D. Exactly one object is serialized.

E. Exactly two objects are serialized.

Answer:

-> A is correct. An instance of type Computer Has-a Keyboard. Because Keyboard doesn't implement Serializable, any attempt to serialize an instance of Computer will cause an exception to be thrown.

-> B, C, D, and E are incorrect based on the above. If Keyboard did implement Serializable then two objects would have been serialized.

20

Given:

import java.io.\*;

public class TestSer {

public static void main(String[] args) {

SpecialSerial s = new SpecialSerial();

try {

ObjectOutputStream os = new ObjectOutputStream(

new FileOutputStream("myFile"));

os.writeObject(s); os.close();

System.out.print(++s.z + " ");

ObjectInputStream is = new ObjectInputStream(

new FileInputStream("myFile"));

SpecialSerial s2 = (SpecialSerial)is.readObject();

is.close();

System.out.println(s2.y + " " + s2.z);

} catch (Exception x) {System.out.println("exc"); }

}

}

class SpecialSerial implements Serializable {

transient int y = 7;

static int z = 9;

}

Which are true? (Choose all that apply.)

A. Compilation fails. B. The output is 10 0 9

C. The output is 10 0 10 D. The output is 10 7 9

E. The output is 10 7 10

F. In order to alter the standard deserialization process you would override the readObject() method in SpecialSerial.

G. In order to alter the standard deserialization process you would override the

defaultReadObject() method in SpecialSerial.

Answer:

-> C and F are correct. C is correct because static and transient variables are not serialized when an object is serialized. F is a valid statement.

-> A, B, D, and E are incorrect based on the above. G is incorrect because you don't

override the defaultReadObject() method, you call it from within the overridden

readObject()method, along with any custom read operations your class needs.

21

Which class is used to read streams of characters from a file? (1 correct answer)

1. FileReader
2. FileWriter
3. FileInputStream
4. FileOutputStream

Resposta: A

22

What happens when this code is compiled and executed? (1 correct answer)

void test() {

FileWriter writer = new FileWriter("fun.log");

writer.write("Hello!");

writer.close();

}

1. A file fun.log is created with the content “Hello!”.
2. A file fun.log is created but it’s empty, because flush() was not called.
3. A runtime exception is thrown because flush() was not called.
4. Compilation fails.

Resposta: A

23

Consider a file fun.log whose first line is “Hello!”. What happens when this code is compiled and executed? (1 correct answer)

void test() throws IOException {

File file = new File("C:/fun.log");

BufferedReader reader = new BufferedReader(file);

System.out.println(reader.readLine());

}

1. A runtime exception is thrown because the file already exists.
2. It prints “Hello!”.
3. It prints “null”.
4. Compilation fails.

Resposta: D

24

The file fun.log has already some content. We want to keep the content of the file and add a new line at the end. Which statement can achieve this? (1 correct answer)

void test() throws IOException {

PrintWriter writer = new PrintWriter("C:/fun.log");

// insert statement here

writer.flush();

writer.close();

}

1. writer.println(“this is a new line”);
2. writer.append(“this a new line”);
3. writer.append(“\nthis is a new line”);
4. None of the above.

Resposta; D

25

What happens when this code is compiled and executed? (1 correct answer)

void test() throws IOException {

for (int index = 1; index <= 2; index++) {

PrintWriter writer = new PrintWriter("/apa");

writer.print("apa");

writer.close();

}

}

1. A file apa is created with content “apa”.
2. A file apa is created with content “apaapa”.
3. Two files are created.
4. An exception is thrown at runtime.

Resposta: A