# Domanda **Giusto**

Which of the following cannot be instantiated directly by the caller using the constructor? (Choose two.)

* ​HashMap
* ​Locale
* ​ResourceBundle**(Giusto)**
* ​Locale.Builder
* ​Properties
* ​DateTimeFormatter**(Giusto)**

Spiegazione

The Locale class has a constructor taking a language code and an optional country code. A Locale.Builder is created only using a constructor. The Properties and HashMap classes are concrete types of Map, so they have constructors. By contrast, a ResourceBundle is an abstract class, and instances are typically obtained by calling the ResourceBundle.getBundle() method, making option B correct. Similarly, DateTimeFormatter is obtained using a static method, making option E correct.

# Domanda  **Giusto**

What is the output of the following?

1. var builder = new StringBuilder("Leaves growing");
2. do {
3. builder.delete(0, 5);
4. } while (builder.length()> 5);
5. System.out.println(builder);

* ​

The code does not compile.

* ​

The code compiles but throws an exception at runtime.

* ​

Leaves growing

* ​

wing

**(Giusto)**

* ​

ing

Spiegazione

On the first iteration through the loop, the first five characters are removed, and builder becomes "s growing". Since there are more than five characters left, the loop iterates again. This time, five more characters are removed, and builder becomes "wing". This matches option C.

# Domanda **Sbagliato**

What is the result of the following?

1. import java.util.\*;
2. public class Museums {
3. public static void main(String[] args) {
4. String[] array = {"Natural History", "Science", "Art"};
5. List<String> museums = Arrays.asList(array);
6. museums.remove(2);
7. System.out.println(museums);
8. }
9. }

* ​

[Natural History, Science, Art]

* ​

The code compiles but throws an exception at runtime.

**(Giusto)**

* ​

The code does not compile.

* ​

[Natural History, Science]

**(Sbagliato)**

Spiegazione

When converting an array to a List, Java uses a fixed‐sized backed list. This means that the list uses an array in the implementation. While changing elements to new values is allowed, adding and removing elements is not.

Fine modulo

Inizio modulo

# Domanda **Giusto**

What is the output of the following when run as  java EchoFirst.java seed flower plant?

1. import java.util.\*;
2. public class EchoFirst {
3. public static void main(String[] args) {
4. Arrays.sort(args);
5. var result = Arrays.binarySearch(args, args[0]);
6. System.out.println(result);
7. }
8. }

* ​

1

* ​

The output is not guaranteed.

* ​

The code compiles but throws an exception at runtime.

* ​

2

* ​

0

**(Giusto)**

* ​

The code does not compile.

Spiegazione

This class is called with three command‐line arguments. First, the array is sorted, which meets the pre‐condition for binary search. At this point, the array contains [flower, plant, seed]. The key is to notice the value of args[0] is now flower rather than seed. Calling a binary search to find the position of flower returns 0, which is the index matching that value. Therefore, the answer is option A.

Fine modulo

Inizio modulo

# Domanda **Giusto**

How many lines of the following declaration contain a compiler error?

1. 1: import java.lang.annotation.\*;
2. 2: @Inherited
3. 3: public @interface Panic {
4. 4: public abstract alert() default 10;
5. 5: public final static int alarm\_volume = 10;
6. 6: String[] type() default {"test"};
7. 7: Long range();
8. 8: abstract boolean silent();
9. 9: }

* ​

Three.

* ​

Two.

**(Giusto)**

* ​

None.

* ​

One.

* ​

Four.

* ​

Five.

Spiegazione

Line 4 is missing a type for the element. If a compatible type, such as int, was added before alert(), it would compile. An annotation element type must be a primitive type, a String, a Class, an enum, another annotation, or an array of any of these types. For this reason, line 6 compiles, and line 7 does not. Since lines 4 and 7 are the only lines that do not compile, option C is correct. Notice that lines 5 and 8 use a lot of extra modifiers, like public and abstract, that are usually applied implicitly.

Fine modulo

Inizio modulo

# Domanda **Giusto**

What is the output of the method that main() calls?

1. public class Hippo {
2. private static void hippo(short num1, short num2) {
3. System.out.println("shorts");
4. }
5. private static void hippo(int… nums) {
6. System.out.println("varargs");
7. }
8. private void hippo(long num1, long num2) {
9. System.out.println("longs");
10. }
11. private void hippo(int num1, int num2) {
12. System.out.println("nums");
13. }
14. public static void main(String… args) {
15. hippo(1, 5);
16. }
17. }

* ​

varargs

**(Giusto)**

* ​

nums

* ​

longs

* ​

shorts

* ​

The code does not compile.

Spiegazione

This code is tricky. Java picks the most specific method signature it can find in an overloading situation. Normally, that would be the one with the two int parameters. However, that method is an instance method, and the main() method is looking for a static method. The next choice is the varargs one, making option D the answer.

Fine modulo

Inizio modulo

# Domanda **Giusto**

Which statements best describe the result of executing this code? (Choose two.)

1. package nyc;
2. public class TouristBus {
3. public static void main(String… args) {
4. var nycTourLoops = new String[] {
5. "Downtown", "Uptown", "Brooklyn" };
6. var times = new String[] { "Day", "Night" };
7. for (int i = 0, j = 0;
8. i < nycTourLoops.length; i++, j++)
9. System.out.println(
10. nycTourLoops[i] + " " + times[j]);
11. }
12. }

* ​

The code throws an exception at runtime.

**(Giusto)**

* ​

The println() causes two lines of output.

**(Giusto)**

* ​

The println() causes three lines of output.

* ​

The code terminates successfully.

* ​

The println() causes one line of output.

Spiegazione

The first two iterations through the loop complete successfully, making option B correct. However, the two arrays are not the same size, and the for loop only checks the size of the first one. The third iteration throws an ArrayIndexOutOfBoundsException, making option E correct.

Fine modulo

Inizio modulo

# Domanda **Giusto**

Which lines fail to compile?

1. package armory;
2. import java.util.function.\*;
3. interface Shield {
4. void protect();
5. }
6. class Dragon {
7. int addDragon(Integer count) {
8. return ++count;
9. }
10. }
11. public class Sword {
12. public static void main(String[] knight) {
13. var dragon = new Dragon();
14. Function<Shield, Sword> func = Shield::protect; // line x
15. UnaryOperator<Integer> op = dragon::addDragon; // line y
16. }
17. }

* ​

The code compiles.

* ​

Both lines x and y

* ​

Only line y

* ​

Only line x

**(Giusto)**

Spiegazione

The method reference on line x is supposed to define a Function. The Shield interface does define a single abstract method. However, that method has a void return type, which is not compatible with Function. Line y does compile since addDragon() has both a parameter and return type. Option A is the answer, since only line x fails to compile.

Fine modulo

Inizio modulo

# Domanda **Giusto**

What is true of the following method?

1. public void printColor() {
2. System.out.println(color);
3. }

* ​

It is a correctly implemented mutator method.

* ​

It is an incorrectly implemented mutator method.

* ​

It is an incorrectly implemented accessor method.

* ​

It is a correctly implemented accessor method.

* ​

None of the above.

**(Giusto)**

Spiegazione

This method does not set or return a value so it is not an accessor or mutator. Therefore, option E is correct.

Fine modulo

Inizio modulo

# Domanda **Sbagliato**

Fill in the blanks: The operators +=, \_\_\_\_\_\_, \_\_\_\_\_\_, \_\_\_\_\_\_\_, \_\_\_\_\_\_\_, and ‐‐ are listed in increasing or the same level of operator precedence. (Choose two.)

* ​

<=, >=, !=, !

* ​

\*, /, %, ++

**(Giusto)**

* ​

=, +, /, \*

**(Giusto)**

* ​

^, \*, =, ++

**(Sbagliato)**

* ​

^, \*, ==, ++

* ​

%, \*, /, &&

Spiegazione

In option A, the assignment operator (=) incorrectly comes after the multiplication (\*) operator. In option B, the shortcircuit logical operator (&&) incorrectly comes after the division (/) operator. In option D, the equality operator (==) incorrectly comes after the multiplication (\*) operator. In option F, the not equals operator (!=) incorrectly comes after the relational operators, (<= and >=). This leaves options C and E as the correct answers. For these answers, it may help to remember that the modulus operator (%), multiplication operator (\*), and division operator (/) have the same operator precedence.

Fine modulo

Inizio modulo

# Domanda **Giusto**

Which of the following are valid functional interfaces? (Choose two.)

* ​
  1. interface CanSwim {
  2. abstract Long swim();
  3. boolean test();
  4. }
* ​
  1. interface CanFly {
  2. abstract void fly();
  3. }

**(Giusto)**

* ​
  1. interface CanRun {
  2. void run();
  3. static double runFaster() {return 2.0; }
  4. }

**(Giusto)**

* ​
  1. interface CanClimb {
  2. default void climb() {}
  3. static void climb(int x) {}
  4. }
* ​
  1. interface CanDance {
  2. int dance() { return 5; }
  3. }

Spiegazione

To be a valid functional interface, an interface must declare exactly one abstract method. Option A is incorrect, because CanClimb does not contain any abstract methods. Next, all interface methods not marked default or static are assumed to be abstract, and abstract methods cannot have a body. For this reason, CanDance does not compile, making option B incorrect. Options C and D are correct answers because each contains exactly one abstract method. Option E is incorrect because it contains two abstract methods, since test() is assumed to be abstract.

Fine modulo

Inizio modulo

# Domanda : **Giusto**

Starting with **DoubleBinaryOperator**and going downward, fill in

the values for the table.

**Functional Interface**            =>                **# Parameters in Method  Signature**

***DoubleBinaryOperator     =***

***LongToIntFunction            =***

***ToLongBiFunction             =***

***IntSupplier                         =***

***ObjLongConsumer            =***

* ​

3, 0, 2, 1, 1

* ​

2, 1, 2, 0, 2

**(Giusto)**

* ​

2, 1, 1, 0, 1

* ​

1, 2, 1, 0, 1

* ​

2, 1, 0, 1, 2

* ​

1, 0, 0, 0, 2

Spiegazione

DoubleBinaryOperator takes two double values and returns a double value. LongToIntFunction takes one long value and returns an int value. ToLongBiFunction takes two generic arguments and returns a long value. IntSupplier does not take any values and returns an int value. ObjLongConsumer takes one generic and one long value and does not return a value. For these reasons, option E is the correct answer.

Fine modulo

Inizio modulo

# Domanda **Sbagliato**

What is the result of compiling and executing the following application?

1. package reptile;
2. public class Alligator {
3. static int teeth;
4. double scaleToughness;
5. public Alligator() {
6. this.teeth++;
7. }
8. public void snap(int teeth) {
9. System.out.print(teeth+" ");
10. teeth--;
11. }
12. public static void main(String[] unused) {
13. new Alligator().snap(teeth);
14. new Alligator().snap(teeth);
15. }
16. }

* ​

0 1

* ​

1 2

**(Giusto)**

* ​

2 2

* ​

1 1

**(Sbagliato)**

* ​

The code compiles but produces an exception at runtime.

* ​

The code does not compile.

Spiegazione

The code compiles and runs without exception, making options E and F incorrect. The question is testing your knowledge of variable scope. The teeth variable is static in the Alligator class, meaning the same value is accessible from all instances of the class, including the static main() method. The static variable teeth is incremented each time the constructor is called. Note that the constructor uses this to access a static variable, which is bad practice, but allowed. Since teeth is a local variable within the snap() method, the argument value is used, but changes to the local variable do not affect the static variable teeth. The local variable teeth is not used after it is decremented, the decrement operation has no meaningful effect on the program flow or the static variable teeth. Since the constructor is called twice, with snap() executed after each constructor call, the output printed is 1 2, making option C the correct answer.

Fine modulo

Inizio modulo

# Domanda **Sbagliato**

Which statements about the following class that loads a library on startup are correct? (Choose three.)

1. import java.io.FilePermission;
2. import java.security.\*;
3. import java.util.List;
4. public class Startup {
5. private static final List<String> files =
6. List.of("my.secret","other.secrets");
7. public void startup(String f) {
8. var perm = new FilePermission(f,"write"); // j1
9. AccessController.checkPermission(perm); // j2
10. AccessController.doPrivileged(
11. new PrivilegedAction<Void>() {
12. public Void run() {
13. if(files.contains(f))
14. System.loadLibrary(f); // j3
15. return null;
16. }
17. });
18. } }

* Line j2 does not contain a security error or risk.**(Giusto)**
* ​Line j1 contains a security error or risk.**(Giusto)**
* ​Line j3 contains a security error or risk.**(Sbagliato)**
* ​Line j1 does not contain a security error or risk.
* ​Line j3 does not contain a security error or risk.**(Giusto)**
* ​Line j2 contains a security error or risk.

Spiegazione

Line j1 should be verifying read or execute access to the file, not write, making option A correct. The system needs permission other than write to load a library. The rest of the lines are valid, making options E and F correct. While the user is free to pass in any value for the file, it is checked against a list of known files, thereby preventing arbitrary file access.

Fine modulo

Inizio modulo

# Domanda: **Sbagliato**

Which of the following statements about performing a concurrent reduction are correct? (Choose two.)

* ​If a collector is used, it must have the concurrent characteristic.**(Giusto)**
* ​The stream must operate on thread‐safe collections.**(Sbagliato)**
* ​If a collector is used, it must have the unordered characteristic.
* ​If the reduce() method is used with a lambda expression, then it should be stateful.
* ​The stream must inherit ParallelStream<T>.
* ​The stream must be parallel.**(Giusto)**

Spiegazione

To perform a concurrent reduction, the stream or the collector must be unordered. Since it is possible to use an ordered collector with an unordered stream and achieve a parallel reduction, option A is incorrect. Option B is also incorrect. While having a thread‐safe collection is preferred, it is not required. Stateful lambda expressions should be avoided, whether the stream is serial or parallel, making option C incorrect. Option D is incorrect as there is no class/interface within the JDK called ParallelStream. Options E and F are correct statements about performing parallel reductions.

Fine modulo

Inizio modulo

# Domanda: **Sbagliato**

Which two options when inserted independently can fill in the blank to compile the code? (Choose two.)

javac \_\_\_\_\_\_ mods -d birds com-bird/\*.java \*.java

* ​‐m
* ​‐p**(Giusto)**
* ​‐‐module‐path**(Giusto)**
* ​‐‐classpath**(Sbagliato)**
* ​‐cp**(Sbagliato)**
* ​‐classpath

Spiegazione

The javac command takes a ‐‐module‐path parameter. You need to memorize that the short form of this option is ‐p. This makes options C and F the answer.

Fine modulo

Inizio modulo

# Domanda : **Sbagliato**

Which of the following is true of the following module declaration?

1. 1: module com.mammal {
2. 2: exports com.mammal.cat;
3. 3: exports com.mammal.mouse to com.mice;
4. 4: uses com.animal;
5. 5: }

* ​The code compiles.**(Giusto)**
* ​The first line that fails to compile is line 3.**(Sbagliato)**
* ​The first line that fails to compile is line 4.
* ​The first line that fails to compile is line 2.
* ​The first line that fails to compile is line 1.

Spiegazione

This is a correct module‐info file. It exports the com.mammal.cat package to any modules that want to use it. By contrast, it exports the com.mammal.mouse package to only one other module. Finally, it demonstrates the uses directive. Since the code is correct, option E is the answer.

Fine modulo

Inizio modulo

# Domanda 18: **Sbagliato**

Which of the following are true about Java operators and statements? (Choose three.)

* ​The logical operators (|) and (||) are interchangeable, producing the same results at runtime.
* ​The complement operator (!) operator may be applied to numeric expressions.
* ​Both right‐hand sides of the ternary expression are evaluated at runtime.

**(Sbagliato)**

* ​The post‐increment operator (++) returns the value of the variable before the addition is applied.

**(Giusto)**

* ​An assignment operator returns a value that is equal to the value of the expression being assigned.

**(Giusto)**

* ​A switch statement may contain at most one default statement.

**(Giusto)**

Spiegazione

The ternary operator (? :) evaluates only one of the two right‐hand expressions at runtime, so option A is incorrect. A switch statement may contain at most one optional default statement, making option B correct. The post‐increment operator increments the variable and returns the original value, making option C correct. The logical operator (|) operator will evaluate both operands, while the disjunctive short‐circuit (||) operator will only evaluate the right‐hand side of the expression if the left‐hand side evaluates to false. Therefore, they may produce different results if the left operand is true, and option D is incorrect. Option E is incorrect as the complement operator (!) is applied to boolean values. Finally, option F is correct and allows the assignment operator to be used in a conditional expression, such as part of a loop condition.

Fine modulo

Inizio modulo

# Domanda 19: **Giusto**

Which of the following methods can run without error for at least one SQL query?

1. private static void choices(PreparedStatement ps,
2. String sql) throws SQLException {
3. try (var rs = ps.executeQuery()) {
4. System.out.println(rs.getInt(1));
5. }
6. }
7. private static void moreChoices(PreparedStatement ps,
8. String sql) throws SQLException {
9. try (var rs = ps.executeQuery()) {
10. rs.next();
11. System.out.println(rs.getInt(1));
12. }
13. }
14. private static void stillMoreChoices(PreparedStatement ps,
15. String sql) throws SQLException {
16. try (var rs = ps.executeQuery()) {
17. if (rs.next())
18. System.out.println(rs.getInt(1));
19. }
20. }
21. }

* ​

moreChoices() and stillMoreChoices()

**(Giusto)**

* ​

moreChoices()

* ​

choices() and stillMoreChoices()

* ​

All three methods

* ​

stillMoreChoices()

* ​

None of the above

Spiegazione

The most common approach is stillMoreChoices(), which works for any SELECT statement that has an int in the first column. If the SELECT statement has a function like count(\*) or sum(\*) in the first column, there will always be a row in the ResultSet, so moreChoices() works as well. Therefore, option D is the answer.

Fine modulo

Inizio modulo

# Domanda 20: **Giusto**

Which are true statements about interfaces and abstract classes? (Choose three.)

* ​

Both abstract classes and interfaces can have public constructors.

* ​

Both abstract classes and interfaces can have abstract methods.

**(Giusto)**

* ​

All methods in abstract classes are public, while interfaces can use various access modifiers for their methods and variables, including private in some cases.

* ​

Unlike abstract classes, interfaces can be marked final.

* ​

Abstract classes offer support for single inheritance, while interfaces offer support for multiple inheritance.

**(Giusto)**

* ​

Interfaces can only extend other interfaces, while abstract classes can extend both abstract and concrete classes.

**(Giusto)**

Spiegazione

Option B is incorrect as abstract classes allow any of the access modifiers. Option D is incorrect because interfaces do not have constructors. Option F is incorrect because neither abstract classes nor interfaces can be marked final. Options A, C, and E are true statements.

Fine modulo

Inizio modulo

# Domanda 21: **Giusto**

Which of the following statements about InputStream and Reader are correct? (Choose two.)

* ​One contains a read() method that returns a byte value, while the other contains a read() method that returns a char value.
* ​They can both be used to read character data.**(Giusto)**
* ​They are both interfaces.
* ​Only one of them contains a flush() method.
* ​Only one of them contains a skip() method.
* ​They are both abstract classes.**(Giusto)**

Spiegazione

First, they are both classes, not interfaces, making option A correct and option D incorrect. Next, while it is more common to use Reader for character data, InputStream and Reader are both capable of reading character data, making option B correct. Option C is incorrect, as both classes contain a read() method that returns an int value. As you may recall from your studies, neither use byte or char so that ‐1 can be returned when the end of the stream is reached without using an existing byte or char value. Option E is incorrect because neither contains a flush() method, while option F is incorrect because they both contain a skip() method.

Fine modulo

Inizio modulo

# Domanda 22: **Sbagliato**

What can fill in the blank so the play() method can be called from all classes in the com.mammal.eland package, but not the com.mammal.gopher package?

1. package com.mammal;
2. public class Enrichment {
3. \_\_\_\_\_ void play() {}
4. }

* ​private
* ​Leave it blank**(Sbagliato)**
* ​public
* ​None of the above**(Giusto)**
* ​protected

Spiegazione

Leaving it blank gives package‐private access. This would be the correct answer if the code we wanted to receive access were in the same package. Since it is not, we would need modules in order to restrict the access and option E is correct.

Fine modulo

Inizio modulo

# Domanda 23: **Sbagliato**

What is the result of the following?

1. import java.util.stream.\*;
2. public class StreamOfStreams {
3. public static void main(String[] args) {
4. var result =
5. Stream.of(getNums(9, 8), getNums(22, 33)) // c1
6. .flatMap(x -> x) // c2
7. .map((a, b) -> a - b) // c3
8. .filter(x -> !x.isEmpty()) // c4
9. .get();
10. System.out.println(result);
11. }
12. private static Stream<Integer> getNums(int num1, int num2) {
13. return Stream.of(num1, num2);
14. }
15. }

* ​

The first compiler error is on line c2.

* ​

The first compiler error is on line c3.

**(Giusto)**

* ​

The first compiler error is on line c1.

* ​

The code compiles and outputs 1.

* ​

The first compiler error is on line c4.

**(Sbagliato)**

* ​

The code compiles and outputs 8.

Spiegazione

Line c1 correctly creates a stream containing two streams. Line c2 uses flatMap() to create a Stream of four Integer objects. The first problem is on line c3, which tries to use the numbers as if they are still pairs. Since we have a Stream<Integer> at that point, the code does not compile, and option E is the answer. Line c4 does not compile either as you can't call a List method on an Integer.

Fine modulo

Inizio modulo

# Domanda 24: **Sbagliato**

Suppose we have a peacocks table with two columns: name and rating. What does the following code output if the table is empty?

1. 10: var url = "jdbc:derby:birds";
2. 11: var sql = "SELECT name FROM peacocks WHERE name = ?";
3. 12: try (var conn = DriverManager.getConnection(url);
4. 13: var stmt = conn.prepareStatement(sql)) {
5. 14:
6. 15: stmt.setString(1, "Feathers");
7. 16:
8. 17: try (var rs = stmt.execute()) {
9. 18: System.out.println(rs.next());
10. 19: }
11. 20: }

* ​The code throws an exception at runtime.
* ​The code does not compile due to lines 12–13.
* ​false

**(Sbagliato)**

* ​The code does not compile due to lines 17–18.

**(Giusto)**

* ​true
* ​The code does not compile due to another line.

Spiegazione

The execute() method returns a boolean, not a ResultSet. This causes a compiler error on line 18, which corresponds to option D.

Fine modulo

Inizio modulo

# Domanda 25: **Sbagliato**

Which classes when inserted into the blank do not allow this code to compile? (Choose two.)

1. import java.io.\*;
2. class Music {
3. void make() throws IOException {
4. throw new UnsupportedOperationException();
5. }
6. }
7. public class Sing extends Music {
8. public void make() throws \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ {
9. System.out.println("do-re-mi-fa-so-la-ti-do");
10. }
11. }

* ​Error**(Sbagliato)**
* ​Throwable**(Giusto)**
* ​FileNotFoundException
* ​RuntimeException
* ​NumberFormatException
* ​Exception**(Giusto)**

Spiegazione

An overriding method cannot declare any new or broader checked exceptions as the overridden method. Option A is permitted because FileNotFoundException is a narrower exception than IOException. Options B, D, and F are permitted because new unchecked exceptions are allowed. Note that IOException is not required to be declared at all in the overriding method. Options C and E are incorrect because they are broader checked exceptions than IOException. Even if you didn't know Throwable was checked, you should have been able to solve this by process of elimination.

Fine modulo

Inizio modulo

# Domanda 26: **Giusto**

Which of the following statements about nested classes are correct? (Choose three.)

* ​An anonymous class can declare that it implements multiple interfaces.
* ​A local class can access all local variables prior to its declaration within a method.
* ​A local class can declare that it implements multiple interfaces.

**(Giusto)**

* ​E. A static nested class can contain static methods.

**(Giusto)**

* ​A member inner class can contain static methods.
* ​All nested classes can contain constant variables.

**(Giusto)**

Spiegazione

Options B, C, and E are valid statements about nested classes. An anonymous class can declare only one supertype, either a class or an interface, making option A incorrect. A member inner class cannot contain static methods, making option B incorrect. A local class can access only final and effectively final local variables, making option F incorrect.

Fine modulo

Inizio modulo

# Domanda 27: **Giusto**

What is the result of the following?

1. 1: import java.util.function.\*;
2. 2: public class Ready {
3. 3: private static double getNumber() {
4. 4: return .007;
5. 5: }
6. 6: public static void main(String[] args) {
7. 7: Supplier<double> s = Ready::getNumber;
8. 8: double d = s.get();
9. 9: System.out.println(d);
10. 10: }
11. 11: }

* ​The code does not compile due to line 8.
* ​The code does not compile for another reason.
* ​0.007
* ​The code does not compile due to line 7.

**(Giusto)**

* ​0

Spiegazione

You can't use generics with a primitive, so it should be Supplier<Double>. This makes option C the answer. If this were fixed, option B would be the answer.

Fine modulo

Inizio modulo

# Domanda 28: **Sbagliato**

What is the output of executing the following code snippet?

1. var e = Executors.newSingleThreadExecutor();
2. Runnable r1 = () -> Stream.of(1,2,3).parallel();
3. Callable r2 = () -> Stream.of(4,5,6).parallel();
4. Future<Stream<Integer>> f1 = e.submit(r1); // x1
5. Future<Stream<Integer>> f2 = e.submit(r2); // x2
6. var r = Stream.of(f1.get(),f2.get())
7. .flatMap(p -> p) // x3
8. .parallelStream() // x4
9. .collect(
10. Collectors.groupingByConcurrent(i -> i%2==0));
11. System.out.print(r.get(false).size()
12. +" "+r.get(true).size());

* ​One of the marked lines (x1, x2, x3, x4) does not compile.
* ​2 4
* ​Two of the marked lines (x1, x2, x3, x4) do not compile.

**(Giusto)**

* ​Three of the marked lines (x1, x2, x3, x4) do not compile.
* ​None of the above.

**(Sbagliato)**

* ​3 3

Spiegazione

Line x1 does not compile because of an assignment and value mismatch. The r1 variable is a Runnable expression. While there is an ExecutorService.submit() that takes a Runnable expression, it returns Future<?>, since the return type is void. This type is incompatible with the Future<Stream> assignment without an explicit cast, leading to a compiler error. Next, line x4 does not compile. The parallelStream() method is found in the Collection interface, not the Stream interface. Due to these two compilation errors, option D is the correct answer.

Fine modulo

Inizio modulo

# Domanda 29: **Sbagliato**

Suppose you have a consumer that calls the lion() method within a Lion service. You have four distinct modules: consumer, service locator, service provider, and service provider interface.

If you add a parameter to the lion() method, how many of the modules require recompilation?

* ​One.

**(Sbagliato)**

* ​Four.
* ​Two.
* ​Zero.
* ​Three.

**(Giusto)**

Spiegazione

The service provider interface clearly needs to be recompiled, since that's where the change occurs. The service provider also needs to be recompiled because it implements the interface. Finally, the consumer needs to be recompiled because it calls the interface. The service locator does not need to be recompiled as it only knows the service provider interface name rather than its method signature. Since three require recompilation, option D is correct.

Fine modulo

Inizio modulo

# Domanda 30: **Sbagliato**

Assume the file system is accessible, /flower/rose.txt exists, and the other two directories /garden and /nursery do not exist.

What is the expected result after executing the following code snippet?

1. Files.createDirectories(Path.of("/garden"));
2. Files.createDirectory(Path.of("/nursery"));
3. Files.move(Path.of("/flower/rose.txt"),
4. Paths.get("/garden"),
5. StandardCopyOption.REPLACE\_EXISTING);
6. Files.move(new File("/garden/rose.txt").toPath(),
7. Paths.get("/nursery"),
8. StandardCopyOption.ATOMIC\_MOVE);

* ​There is a file at /flower/rose.txt
* ​

The first move() statement throws an exception.

* ​The code does not compile.

**(Sbagliato)**

* ​The second move() statement throws an exception.

**(Giusto)**

* ​None of the above.
* ​There is a file at /nursery/rose.txt

Spiegazione

The code compiles, so option C is incorrect. The first two lines successfully create directories. The first move() statement moves a file from /flower/rose.txt to /garden, not /garden/rose.txt. There is already an empty directory there, but since the REPLACE\_EXISTING flag is provided, the /garden directory is replaced with a file. The next move() statement throws an exception because there is no source file at /garden/rose.txt. For this reason, option E is correct.

Fine modulo

Inizio modulo

Domanda 31: **Giusto**

How many times does this code print [2, 7, 8]?

1. 1: import java.util.\*;
2. 2: import java.util.stream.\*;
3. 3:
4. 4: public class RemoveMe<T> {
5. 5: private List<T> values;
6. 6: public RemoveMe(T… values) {
7. 7: this.values = Arrays.stream(values)
8. 8: .collect(Collectors.toList());
9. 9: }
10. 10: public void remove(T value) {
11. 11: values.remove(value);
12. 12: }
13. 13: public static void main(String[] args) {
14. 14: var integer = new RemoveMe<Integer>(2, 7, 1, 8);
15. 15: var longs = new RemoveMe<Long>(2L, 7L, 1L, 8L);
16. 16: integer.remove(1);
17. 17: longs.remove(1L);
18. 18:
19. 19: System.out.println(integer.values);
20. 20: System.out.println(longs.values);
21. 21:
22. 22: var values = new ArrayList<Integer>();
23. 23: values.add(2);
24. 24: values.add(7);
25. 25: values.add(1);
26. 26: values.add(8);
27. 27: values.remove(1);
28. 28: System.out.println(values);
29. 29: }
30. 30: }

* ​

One.

* ​

The code compiles but throws an exception.

* ​

Three.

* ​

The code does not compile.

* ​

Zero.

* ​

Two.

**(Giusto)**

Spiegazione

Lines 14 and 15 create RemoveMe<Integer> and RemoveMe<Long> instances, respectively. Since we are using generics, the method calls on lines 16 and 17 autobox the primitive values (1 and 1L) to the Integer value 1 and Long value 1L, respectively. Therefore, the method on line 11 removes the argument that matches this object value, and not the element at index 1, from the two lists. These are the two lines that print [2, 7, 8]. Line 27 is trickier. Since we are passing a primitive int, the index is used (and not an object matching the value) in the call to remove().This means line 28 prints [2, 1, 8], which is not what we are looking for. Since only lines 16 and 17 give us the desired output, option C is the answer.

Fine modulo

Inizio modulo

Domanda 32: **Sbagliato**

Which commands can include the following output? (Choose two.)

1. JDK Internal API Suggested Replacement
2. sun.misc.Unsafe See http://openjdk.java.net/jeps/260

* ​

jdeps ‐jdkinternals sneaky.jar

**(Giusto)**

* ​

jdeps ‐‐jdk‐internals sneaky.jar

**(Giusto)**

* ​

jdeps –s sneaky.jar

* ​

jdeps –j sneaky.jar

**(Sbagliato)**

* ​

jdeps ‐‐internals sneaky.jar

* ​

jdeps sneaky.jar

Spiegazione

Options B and D are incorrect because those flags do not exist on the jdeps command. Options A and C do exist, but do not include suggested replacements. Options E and F are correct as they will include a table of suggestions if any internal APIs are used in the JAR.

Fine modulo

Inizio modulo

Domanda 33: **Sbagliato**

Fill in the blanks with the proper method names to deserialize an object. (Choose two.)

1. import java.io.\*;
2. public class BoxOfSecrets {
3. private void \_\_\_\_\_\_\_\_\_\_\_\_(ObjectInputStream in)
4. throws IOException {
5. // IMPLEMENTATION OMITTED
6. }
7. public Object \_\_\_\_\_\_\_\_\_\_\_\_() throws
8. ObjectStreamException {
9. // IMPLEMENTATION OMITTED
10. }
11. }

* ​

writeResolve in the first blank

* ​

readResolve in the second blank

**(Giusto)**

* ​

writeObject in the first blank

**(Sbagliato)**

* ​

readReplace in the second blank

* ​

writeReplace in the second blank

* ​

readObject in the first blank

**(Giusto)**

Spiegazione

The write methods are used as part of serialization, not deserialization, making options A, B, and D incorrect. Option C and E are correct because they use the correct method parameters and return types for readObject() and readResolve(). The method names used in options B and F, writeResolve() and readReplace() respectively, are not actually serialization or deserialization methods.

Fine modulo

Inizio modulo

Domanda 34: **Giusto**

How many lines of the following application do not compile?

1. 1: package castles;
2. 2: import java.io.\*;
3. 3: public class Palace {
4. 4: public void openDrawbridge() throws Exception {
5. 5: try {
6. 6: throw new Exception("Problem");
7. 7: } catch (IOException e) {
8. 8: throw new IOException();
9. 9: } catch (FileNotFoundException e) {
10. 10: try {
11. 11: throw new IOException();
12. 12: } catch (Exception e) {
13. 13: } finally {
14. 14: System.out.println("Almost done");
15. 15: }
16. 16: } finally {
17. 17: throw new RuntimeException("Unending problem");
18. 18: }
19. 19: }
20. 20:
21. 21: public static void main(String[] moat)
22. 22: throws IllegalArgumentException {
23. 23: new Palace().openDrawbridge();
24. 24: }
25. 25: }

* ​

One.

* ​

Two.

* ​

Five.

* ​

Three.

**(Giusto)**

* ​

None. The code compiles and produces a stack trace at runtime.

* ​

Four.

Spiegazione

The second catch block in openDrawbridge() is unreachable since FileNotFoundException is a subclass of IOException. The catch blocks should be ordered with the narrower exception classes before the broader ones. For this reason, line 9 does not compile. Next, the local variable e is declared twice within the same scope, with the declaration on line 12 failing to compile. Finally, the openDrawbridge() method declares the checked Exception class, but it is not handled in the main() method on line 23. Since lines 9, 12, and 23 do not compile, option D is correct.

Fine modulo

Inizio modulo

Domanda 35: Saltato

Which of the following sequences can fill in the blanks so the code prints ‐1 0 2?

1. char[][] letters = new char[][] {
2. new char[] { 'a', 'e', 'i', 'o', 'u'},
3. new char[] { 'a', 'e', 'o', 'u'} };
4. var x = Arrays.\_\_\_\_\_\_\_\_\_\_\_\_(letters[0], letters[0]);
5. var y = Arrays.\_\_\_\_\_\_\_\_\_\_\_\_(letters[0], letters[0]);
6. var z = Arrays.\_\_\_\_\_\_\_\_\_\_\_\_(letters[0], letters[1]);
7. System.out.print(x + " " + y + " " + z);

* ​compare, mismatch, mismatch
* ​mismatch, compare, compare
* ​compare, mismatch, compare
* ​mismatch, compare, mismatch**(Giusto)**
* ​None of the above

Spiegazione

When the arrays are the same, the compare() method returns 0, while the mismatch() method returns ‐1. This narrows it down to option C or option D. When the arrays are different, mismatch() returns the index of the first element that is different. In our case, this is index 2, making option D correct. By contrast, the compare() method would return a negative number if filling in the third blank since 'i' is smaller than 'o'.

public static int mismatch​(boolean[] a, boolean[] b)

Finds and returns the index of the first mismatch between two boolean arrays, otherwise return -1 if no mismatch is found. The index will be in the range of 0 (inclusive) up to the length (inclusive) of the smaller array.

Domanda 36: **Giusto**

Which statements about the following class are correct? (Choose two.)

1. package knowledge;
2. class InformationException extends Exception {}
3. public class LackOfInformationException
4. extends InformationException {
5. public LackOfInformationException() { // t1
6. super("");
7. }
8. public LackOfInformationException(String s) { // t2
9. this(new Exception(s));
10. }
11. public LackOfInformationException(Exception c) { // t3
12. super();
13. }
14. @Override public String getMessage() {
15. return "lackOf";
16. }
17. }

* ​LackOfInformationException is a checked exception. **(Giusto)**
* ​The constructor declared at line t2 does not compile.
* ​ The constructor declared at line t1 does not compile.**(Giusto)**
* ​The getMessage() method does not compile because of the @Override annotation.
* ​The constructor declared at line t3 does not compile.
* ​LackOfInformationException compiles without issue.

Spiegazione

The LackOfInformationException class does not compile, making option A incorrect. The compiler inserts the default noargument constructor into InformationException, since the class does not explicitly define any. Since LackOfInformationException extends InformationException, the only constructor available in the parent class is the no‐argument call to super(). For this reason, the constructor defined at line t1 does not compile because it calls a nonexistent parent constructor that takes a String value, and option B is one of the correct answers. The other two constructors at lines t2 and t3 compile without issue, making options C and D incorrect. Option E is also incorrect. The getMessage() method is inherited, so applying the @Override annotation is allowed by the compiler. Option F is the other correct answer. The LackOfInformationException is a checked exception because it inherits Exception but not RuntimeException.

Class: 6

Domanda 37: **Sbagliato**

Assuming the following class is concurrently accessed by numerous threads, which statement about the CountSheep class is correct?

1. package fence;
2. import java.util.concurrent.atomic.\*;
3. public class CountSheep {
4. private static AtomicInteger counter = new
5. AtomicInteger();
6. private Object lock = new Object();
7. public synchronized int increment1() {
8. return counter.incrementAndGet();
9. }
10. public static synchronized int increment2() {
11. return counter.getAndIncrement();
12. }
13. public int increment3() {
14. synchronized(lock) {
15. return counter.getAndIncrement();
16. } } }

* The class does not compile.
* The class compiles but may throw an exception at runtime.

**(Sbagliato)**

* ​The class is already thread‐safe.**(Giusto)**
* ​The class is thread‐safe only if increment2() is removed.
* ​The class is thread‐safe only if increment1() is removed.
* ​The class is thread‐safe only if increment3() is removed.

Spiegazione

The code compiles, making option E incorrect. The key here is that the AtomicInteger variable is thread‐safe regardless of the synchronization methods used to access it. Therefore, synchronizing on an instance object, as in increment1() or increment3(), or on the class object, as in increment2(), is unnecessary because the AtomicInteger class is already threadsafe. For this reason, option D is the correct answer.

Class:7

Domanda 38: **Giusto**

What does the following output?

1. var dice = new LinkedList<Integer>();
2. dice.offer(3);
3. dice.offer(2);
4. dice.offer(4);
5. System.out.print(dice.stream().filter(n -> n != 4));

* [3 2]
* 3
* The code does not compile.
* 2
* None of the above.**(Giusto)**

Spiegazione

The code correctly creates a LinkedList with three elements. The stream pipeline does compile. However, there is no terminal operation, which means the stream is never evaluated, and the output is something like java.util.stream.ReferencePipeline$2@404b9385. This is definitely not one of the listed choices, so option E is correct.

Class: 4

Fine modulo

Inizio modulo

Domanda 39: **Sbagliato**

What is the output of the following application?

1. package woods;
2. interface Plant {
3. default String grow() { return "Grow!"; }
4. }
5. interface Living {
6. public default String grow() { return "Super Growing!"; }
7. }
8. public class Tree implements Plant, Living { // m1
9. public String grow() { return super.Plant.grow(); }
10. public static void main(String[] leaves) {
11. Plant p = new Tree(); // m2
12. System.out.print(((Living)p).grow()); // m3
13. }
14. }

* ​Grow!
* Super Growing!
* It does not compile because of line m2.
* It does not compile because of line m3.
* None of the above.**(Giusto)**
* ​It does not compile because of line m1.**(Sbagliato)**

Spiegazione

A class can inherit two default interfaces with the same signature, so long as it correctly overrides them, which Tree does.

It can also call an inherited version of the default method within an instance method, provided it uses the proper syntax.

In this case, it does not. The correct syntax is Plant.super.grow(), not super.Plant.grow().

NO, the correct syntax is new Tree().grow()

For this reason, this line does not compile. Since it is the only line that does not compile, option F is correct.

Class: 1

Fine modulo

Inizio modulo

Domanda 40: **Sbagliato**

Given the following classes, what is the output of the Watch program?

1. 1: class SmartWatch extends Watch {
2. 2: private String getType() { return "smart watch";
3. }
4. 3: public String getName() {
5. 4: return getType() + ",";
6. 5: }
7. 6: }
8. 7: public class Watch {
9. 8: private String getType() { return "watch"; }
10. 9: public String getName(String suffix) {
11. 10: return getType() + suffix;
12. 11: }
13. 12: public static void main(String[] args) {
14. 13: Watch watch = new Watch();
15. 14: Watch smartWatch = new SmartWatch();
16. 15: System.out.print(watch.getName(","));
17. 16: System.out.print(smartWatch.getName(""));
18. 17: }
19. 18: }

* ​None of the above.
* smart watch,watch
* The code does not compile.
* smart watch,smart watch
* watch,smart watch**(Sbagliato)**
* watch,watch**(Giusto)**

Spiegazione

Line 15 calls the method on line 9 since it is a Watch object, printing watch. Line 16 is a SmartWatch object. However, the getName() method is not overridden in SmartWatch, since the method signature is different. Therefore, the method on line 9 gets called again. That method calls getType(). Since this is a private method, it is not overridden, and watch is printed again, making option D the correct answer.

Class: 1

Fine modulo

Inizio modulo

Domanda 41: **Sbagliato**

What is the output of the following application?

1. 1: package fruit;
2. 2: enum Season {
3. 3: SPRING(1), SUMMER(2), FALL(3), WINTER(4);
4. 4: public Season(int orderId) {}
5. 5: }
6. 6: public class PickApples {
7. 7: public static void main(String… orchard) {
8. 8: final Season s = Season.FALL;
9. 9: switch(s) {
10. 10: case Season.FALL:
11. 11: System.out.println("Time to pick!");
12. 12: default:
13. 13: System.out.println("Not yet!");
14. 14: }
15. 15: }
16. 16: }

* ​Time to pick!
* ​Two lines of code do not compile.**(Giusto)**
* ​The code compiles but prints an exception at runtime.
* ​One line of code does not compile.**(Sbagliato)**
* ​Three lines of code do not compile.
* ​Time to pick! followed by Not yet!

Spiegazione

Line 4 does not compile because enum constructors cannot be public. Line 10 also does not compile because a case statement must use an enum value without the type. In particular, FALL is permitted, but Season.FALL is not. For these two reasons, option D is correct.

Fine modulo

Class: 1

Inizio modulo

Domanda 42: **Giusto**

How many of the following lines contain a compiler error?

1. long min1= 123.0, max1 = 987L;
2. final long min2 = 1\_2\_3, max2 = 9\_\_8\_\_7;
3. long min3 = 123, int max3 = 987;
4. long min4 = 123L, max4 = 987;
5. long min5 = 123\_, max5 = \_987;

* Four
* Two
* One
* Five
* Three**(Giusto)**
* ​Zero

Spiegazione

A long cannot contain a number with decimal points, preventing min1 from compiling. When declaring multiple variables in the same statement, the type is only declared once. Therefore, max3 does not compile.

Underscores in numeric expressions are allowed as long as they are between two digits, making the line with min5 and max5 incorrect. Since three lines have compiler errors, the answer is option D. The L suffix is valid, as is having multiple underscores in a row.

Class: 1

Fine modulo

Inizio modulo

Domanda 43: **Sbagliato**

Which of the following statements about java.lang.Error are most accurate? (Choose two.)

* An Error should be thrown if a file system resource becomes temporarily unavailable.**(Sbagliato)**
* Error is a subclass of Exception, making it a checked exception.
* An application should never catch an Error.**(Giusto)**
* An Error should be thrown if a user enters invalid input.
* Error is a subclass of RuntimeException, making it an unchecked exception.
* ​It is possible to catch and handle an Error thrown in an application.

**(Giusto)**

Spiegazione

An Error indicates an unrecoverable problem.

Options A and E are incorrect because the application could possibly recover.

While it is possible to catch an Error, it is strongly recommended that an application never do so, making options B and D correct.

Finally, options C and F are incorrect because Error extends from Throwable, not Exception or RuntimeException, although it is an unchecked exception.

Class: 6

Fine modulo

Inizio modulo

Domanda 44: **Sbagliato**

What is the output of the following?

1. public class Legos {
2. public static void main(String[] args) {
3. var ok = true;
4. if (ok) {
5. StringBuilder sb = new StringBuilder();
6. sb.append("red");
7. sb.deleteCharAt(0);
8. sb.delete(1, 1);
9. }
10. System.out.println(sb);
11. }
12. }

* ​The code compiles but throws an exception at runtime.
* ed
* The code does not compile.**(Giusto)**
* ​e**(Sbagliato)**
* ​red
* ​r

Spiegazione

This code does not compile because the println() attempts to reference the sb variable. However, that variable is only in scope for the if statement. Since the code does not compile, option E is correct. If the println() were inside the if statement, option C would be correct.

Fine modulo

Class: 3

Inizio modulo

Domanda 45: **Sbagliato**

Which annotations will trigger a compiler error if incorrectly applied to a method with no other annotations? (Choose three.)

​@SuppressWarnings("unchecked")

@Documented**(Giusto)**

​@Override**(Giusto)**

F @SafeVarargs **(Giusto)**

@Deprecated

@SuppressWarnings("magic")**(Sbagliato)**

Spiegazione

@**Documented** can be applied only to annotations, not methods, making option A correct.

If @**Override** is applied to a method that is not actually overridden, a compiler error will ensue, making option D correct.

The @**SafeVarargs** annotation will trigger a compiler error if applied to a method without a vararg parameter or without a final, private, or static modifier, making option F correct.

The rest of the annotations can be applied to methods without triggering a compiler error. For option E, the compiler might not recognize the cause (such as magic), but it will still compile.

Class: 11

Fine modulo

Inizio modulo

Domanda 46: **Sbagliato**

Given that FileNotFoundException is a subclass of IOException and Long is a subclass of Number, what is the output of the following application?

1. package materials;
2. import java.io.\*;
3. class CarbonStructure {
4. protected long count;
5. public abstract Number getCount() throws
6. IOException; // q1
7. public CarbonStructure(int count) { this.count = count; }
8. }
9. public class Diamond extends CarbonStructure {
10. public Diamond() { super(15); }
11. public Long getCount() throws FileNotFoundException
12. { // q2
13. return count;
14. }
15. public static void main(String[] cost) {
16. try {
17. final CarbonStructure ring = new Diamond(); // q3
18. System.out.print(ring.getCount()); // q4
19. } catch (IOException e) {
20. e.printStackTrace();
21. }
22. }
23. }

* The class compiles but produces an exception at runtime.
* It does not compile because of line q2.
* It does not compile because of line q3.
* It does not compile because of line q1.**(Giusto)**
* ​It does not compile because of line q4.
* ​15**(Sbagliato)**

Spiegazione

This problem appears to be about overriding a method, but in fact, it is much simpler. The class CarbonStructure is not declared abstract, yet it includes an abstract method. To fix it, the definition of CarbonStructure would have to be changed to be an abstract class, or the abstract modifier would need to be removed from getCount() in CarbonStructure and a method body added. Since the only answer choice available is to change the getCount() method on line q1, option B is the correct answer. Note that the rest of the application, including the override on line q2, is correct and compiles without issue. The return types Long and Number are covariant since Number is a superclass of Long. Likewise, the exception thrown in the subclass method is narrower, so no compilation error occurs on

q2.

Class: 1

Fine modulo

Inizio modulo

Domanda 47: **Sbagliato**

Which are true statements about the majority of steps in migrating to a modular application? (Choose two.)

D ​In a top‐down migration, automatic modules turn into named modules. **(Giusto)**

C ​In a bottom‐up migration, unnamed modules turn into named modules. **(Giusto)**

F ​In a top‐down migration, unnamed modules turn into named modules.**(Sbagliato)**

A ​In a bottom‐up migration, automatic modules turn into named modules.**(Sbagliato)**

B ​In a bottom‐up migration, named modules turn into automatic modules

E ​In a top‐down migration, named modules turn into automatic modules.

Spiegazione

A fully modular application has all named modules, making options B and E incorrect.

A bottom‐up migration starts out with unnamed modules, making option C correct.

By contrast, a top‐down migration starts by making all modules automatic modules, making option D correct.

Quindi 2 migrazioni possibili to a modular application:

1. Botton-up: da unnamed to named
2. Top-down: making all modules automatic and after named.

Class: 11

Fine modulo

Inizio modulo

Domanda 48: **Sbagliato**

Which of the following are valid in a Java file, listed in the order in which they are declared? (Choose two.)

* ​

A package statement, 128 import statements, and two public class declarations

* ​

A package‐private class declaration and a public interface declaration

**(Giusto)**

* ​

A private interface

**(Sbagliato)**

* ​

16 import statements, a package statement, and a public class declaration

* ​

Two package statements, an import statement, and a public interface declaration

* ​

5 import statements and 7 package‐private interface declarations

**(Giusto)**

Spiegazione

A Java file can have at most one top‐level type and any number of package‐private types. For this reason, options A and E are correct. Option B is incorrect because it can have at most one package statement. Option C is incorrect because it cannot have two public top‐level classes. Option D is incorrect because the package statement must appear before the import statement. Option F is incorrect because top‐level types may not be private or protected.

Fine modulo

Inizio modulo

Domanda 49: **Giusto**

Which statements about the following application are true? (Choose two.)

1. package party;
2. import java.util.concurrent.\*;
3. public class Plan {
4. ExecutorService s =
5. Executors.newScheduledThreadPool(10);
6. public void planEvents() {
7. Runnable r1 = () -> System.out.print("Check food");
8. Runnable r2 = () -> System.out.print("Check drinks");
9. Runnable r3 = () -> System.out.print("Take out trash");
10. s.scheduleWithFixedDelay(r1,1,TimeUnit.HOURS); // g1
11. s.scheduleAtFixedRate(r2,1,1000,TimeUnit.SECONDS); // g2
12. s.execute(r3); // g3
13. s.shutdownNow();
14. } }

* ​

The code hangs indefinitely at runtime.

* ​

Line g1 does not compile.

**(Giusto)**

* ​

Line g3 does not compile.

* ​

The code throws an exception at runtime.

* ​

All of the lines of code compile.

* ​

Line g2 does not compile.

**(Giusto)**

Spiegazione

Lines g1 and g2 do not compile because these methods are available only in the ScheduledExecutorService interface. Since s is of type ExecutorService, the lines referenced in options A and B do not compile. Even if the correct reference type for s was used, line g1 would still fail to compile because scheduleWithFixedDelay() requires two long values, one for the initial delay and one for the period. Line g3 compiles without issue because this method is available in the ExecutorService interface.

Fine modulo

Inizio modulo

Domanda 50: **Giusto**

Which of the following is a valid method name in Java? (Choose two.)

* ​

\_\_\_\_\_\_\_\_

**(Giusto)**

* ​

check‐Activity

* ​

%run

* ​

$Hum2

**(Giusto)**

* ​

sing\\3

* ​

po#ut

Spiegazione

Java methods must start with a letter, the dollar $ symbol, or the underscore \_ character. For this reason, option B is incorrect, and options A and D are correct. Despite how option A looks, it is a valid method name in Java. Options C, E, and F do not compile because the symbols ‐, \, and # are not allowed in method names, respectively.

Fine modulo