Datoteke u Javi

AUDITORNE VJEŽBE

Sadržaj

Primjer rada s datotekama i mapama

```
Scanner input = new Scanner(System.in);

System.out.println("Unesite naziv datoteke ili mape:");

Path path = Paths.get(input.nextLine());

if (Files.exists(path))
{
   // navedena datoteka ili mapa postoji
}
else {
   System.out.printf("%s ne postoji!%n", path);
}
```

Primjer rada s datotekama i mapama

```
System.out.printf("%n%s postoji%n", path.getFileName());
System.out.printf("%s mapa %n", Files.isDirectory(path) ? "je" : "nije");
System.out.printf("%s apsolutna putanja %n", path.isAbsolute() ? "je" : "nije");
System.out.printf("Posljednja promjena: %s%n", Files.getLastModifiedTime(path));
System.out.printf("Veličina: %s%n", Files.size(path));
System.out.printf("Putanja: %s%n", path);
System.out.printf("Apsolutna putanja: %s%n", path.toAbsolutePath());

if (Files.isDirectory(path)) {
      System.out.printf("%nSadržaj mape:%n");

      DirectoryStream<Path> directoryStream = Files.newDirectoryStream(path);

      for (Path p : directoryStream)
         System.out.println(p);
}
```

Primjer rada s datotekama i mapama

- Pomoću statičke metode **Paths.get** pretvara se putanja datoteke ili mape u objekt **Path**
- Metoda **Files.exists** provjerava postoji li datoteka na određenoj putanji
- Metoda **fileName** služi za dohvaćanje naziva datoteke ili mape
- Korištenjem metode **isDirectory** moguće je odrediti je li riječ o mapi ili datoteci, **isAbsolute** provjerava radi li se o apsolutnoj ili relativnoj putanji, a **toAbsolutePath** pretvara putanju u apsolutnu putanju
- Statičkom metodom **getLastModifiedTime** moguće je saznati vrijeme posljednje promjene u datoteci, a metodom **size** dohvaća se veličina datoteke
- Klasom **DirectoryStream** moguće je dohvatiti sadržaj mape

Primjer zapisivanja u datoteku korištenjem Formatter klase

```
private static Formatter output;

public static void main(String[] args) {
   openFile();
   addRecords();
   closeFile();
}


public static void closeFile() {
   if (output != null)
      output.close();
}
```

Primjer zapisivanja u datoteku korištenjem Formatter klase

```
public static void openFile()
  {
     try
     {
        output = new Formatter("clients.txt");
     }
     catch (SecurityException securityException)
     {
        System.err.println("Write permission denied. Terminating.");
        System.exit(1);
     }
     catch (FileNotFoundException fileNotFoundException)
     {
        System.err.println("Error opening file. Terminating.");
        System.exit(1);
  }
}
```

Primjer zapisivanja u datoteku korištenjem Formatter klase

```
public static void addRecords()
   {
      Scanner input = new Scanner(System.in);
      System.out.printf("%s%n%s%n? ",
         "Enter account number, first name, last name and balance.",
         "Enter end-of-file indicator to end input.");
      while (input.hasNext())
      {
         try
         {
           output.format("%d %s %s %.2f%n", input.nextInt(),
               input.next(), input.next(), input.nextDouble());
         }
         catch (FormatterClosedException formatterClosedException)
         {
            System.err.println("Error writing to file. Terminating.");
            break;
         }
         catch (NoSuchElementException elementException)
         {
            System.err.println("Invalid input. Please try again.");
            input.nextLine();
         }

         System.out.print("? ");
      }
   }
```

Primjer čitanja binarne datoteke

```
public static final String FILENAME = "datumi.dat";
public static final int DATE_FORMAT_LENGTH = "dd.MM.yyyy.".length();

public static void main(String[] args) {
        try {
                InputStream in = new FileInputStream(FILENAME);
                char[] data = new char[DATE_FORMAT_LENGTH];
                for (int i = 0; i < data.length; i++) {
                        int datum = in.read();
                        if (datum == -1)
                                break;
                        data[i] = (char) datum;
                }
                System.out.println("Pročitani datum : " + String.valueOf(data));
                in.close();
        } catch (IOException ex) {
                System.err.println(ex.getMessage());
        }
}
```

Primjer kopiranja binarne datoteke

```
public static final String IN_BIN_FILE_NAME = "binary_digits.dat";
public static final String OUT_BIN_FILE_NAME = "binary_digits_copy.dat";

File inFile = new File(IN_BIN_FILE_NAME);
File outFile = new File(OUT_BIN_FILE_NAME);

try (FileInputStream fin = new FileInputStream(inFile);
FileOutputStream fout = new FileOutputStream(outFile)) {
   byte[] buffer = new byte[1024];
   while (true) {
     int bytesRead = fin.read(buffer);
     if (bytesRead == -1)
         break;
     fout.write(buffer, 0, bytesRead);
   }
}
catch(IOException ex) {
   System.err.println(ex.getMessage());
}
```

Primjer kopiranja binarne datoteke s Files klasom

```
public static void copyFile(File from, File to) throws IOException {
    Files.copy(from.toPath(), to.toPath());
}
```

Primjer čitanja tekstualne datoteke

```java
public static final String FILE_NAME = "input.txt";

try (BufferedReader in = new BufferedReader(new FileReader(FILE_NAME))) {
   String line;
   while ((line = in.readLine()) != null) {
      System.out.println(line);
   }
} catch (IOException e) {
   System.err.println(e);
}
```

Primjer čitanja tekstualne datoteke u Javi 11

```
Path datoteka = Path.of("datoteka.txt");

try {
   String tekst = Files.readString(datoteka);
   System.out.println("Pročitana datoteka:");
   System.out.println(tekst);
} catch (IOException e) {
   e.printStackTrace();
}
```

Primjer zapisivanja u tekstualne datoteke

```
public static final String FILE_NAME = "output.txt";

public static void main(String[] args) {
        try (PrintWriter out = new PrintWriter(
                        new FileWriter(new File(FILE_NAME)))) {
                int i = 0;
                do {
                        out.println((i + 1) + ". redak");
                        i++;
                }
                while (i < 10);
        } catch (IOException e) {
                System.err.println(e);
        }
}
```

Primjer zapisivanja u tekstualne datoteke u Javi 11

```
String tekst = "Ovo je tekst koji se zapisuje u datoteku!\nDrugi redak teksta";

Path datoteka = Path.of("datoteka.txt");

try {
   Files.writeString(datoteka, tekst);
} catch (IOException e) {
   e.printStackTrace();
}
```

Primjer korištenja *streamova* s datotekama

```
try (Stream<String> stream = Files.lines(new File("dat/radneMemorije.txt").toPath())) {
   listaStringova = stream.collect(Collectors.toList());
} catch (IOException e) {
   e.printStackTrace();
   logger.error("Došlo je do pogreške u čitanju datoteke!", e);
}
```

- Obvezno koristiti „try-with-resources” kako bi se datoteka automatski zatvorila nakon završetka operacije čitanja

Primjer korištenja *streamova* s datotekama

```
Files.list(new File(".").toPath())
     .filter(p -> !p.getFileName()
                    .toString().startsWith("."))
     .limit(3)
     .forEach(System.out::println);


Files.walk(new File(".").toPath())
     .filter(p -> !p.getFileName()
                   .toString().startsWith("."))
     .forEach(System.out::println);
```

Primjer serijalizacije i deserijalizacije

```
public static final String SERIALIZATION_FILE_NAME = "osobe.dat";

try (ObjectOutputStream out = new ObjectOutputStream(
   new FileOutputStream(SERIALIZATION_FILE_NAME))) {

        Osoba osoba = new Osoba("Pero", "Perić",
               Zupanija.ZUPANIJA_ZAGREBACKA, new Date());

        out.writeObject(osoba);
} catch (IOException ex) {
        System.err.println(ex);
}
```

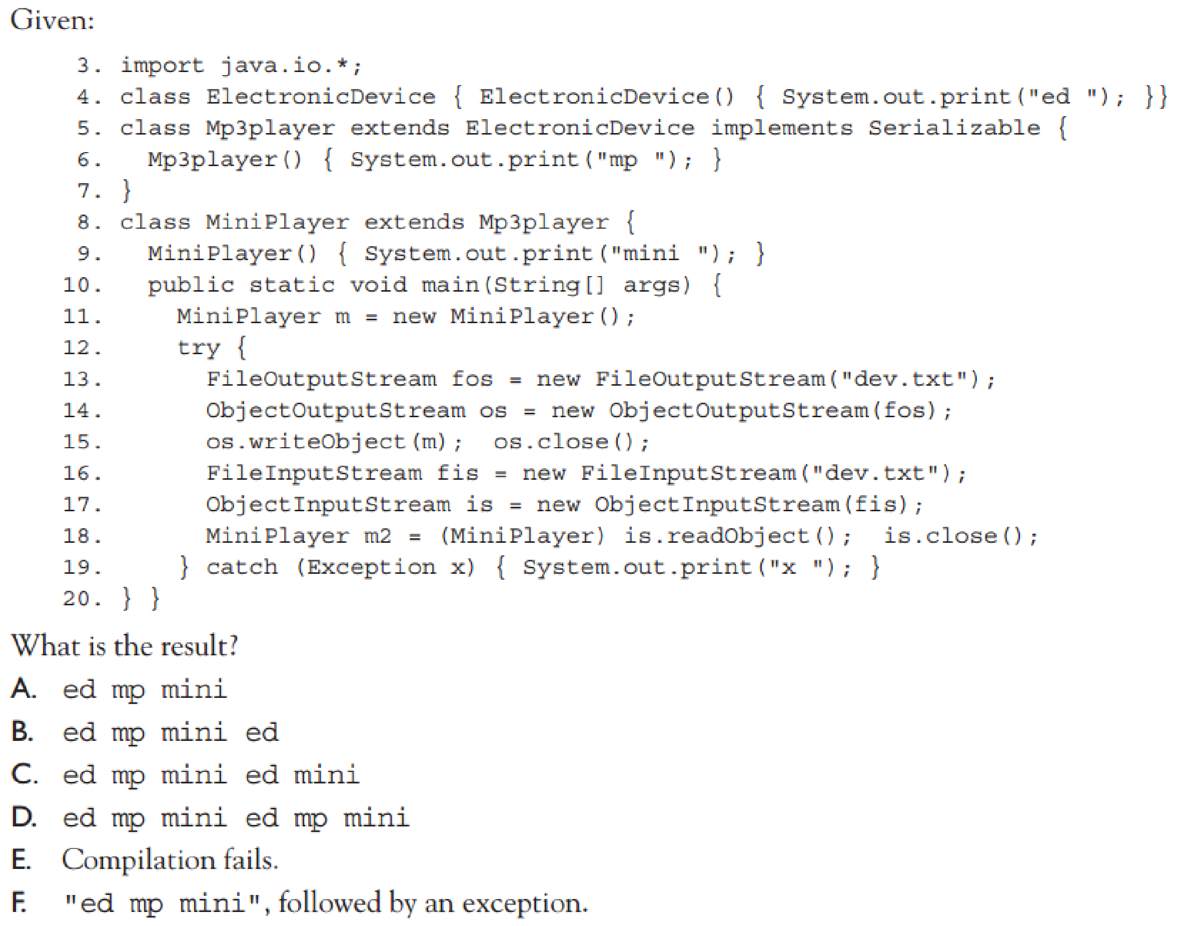
Primjer serijalizacije i deserijalizacije

```java
try (ObjectInputStream in = new ObjectInputStream(
   new FileInputStream(SerijalizacijaTest.SERIALIZATION_FILE_NAME))) {

   Osoba procitanaOsoba = (Osoba) in.readObject();

   System.out.println("Podaci o pročitanom objektu:");
   System.out.println("Ime osobe: " + procitanaOsoba.getIme());
   System.out.println("Prezime osobe: " + procitanaOsoba.getPrezime());
   System.out.println("Datum rođenja osobe: " +
      procitanaOsoba.getDatumRodjenja());
   System.out.println("Županija: " +
      procitanaOsoba.getZupanija().getNaziv());
} catch (IOException ex) {
   System.err.println(ex);
} catch (ClassNotFoundException ex) {
   System.err.println(ex);
}
```

Pitanja s certifikata (1)


Given:

```
import java.io.*;
class ElectronicDevice { ElectronicDevice() { System.out.print("ed "); }}
class Mp3player extends ElectronicDevice implements Serializable {
  Mp3player() { System.out.print("mp "); }
}
class MiniPlayer extends Mp3player {
  MiniPlayer() { System.out.print("mini "); }
  public static void main(String[] args) {
    MiniPlayer m = new MiniPlayer();
    try {
      FileOutputStream fos = new FileOutputStream("dev.txt");
      ObjectOutputStream os = new ObjectOutputStream(fos);
      os.writeObject(m);  os.close();
      FileInputStream fis = new FileInputStream("dev.txt");
      ObjectInputStream is = new ObjectInputStream(fis);
      MiniPlayer m2 = (MiniPlayer) is.readObject();  is.close();
    } catch (Exception x) { System.out.print("x "); }
} }
```

What is the result?

A. `ed mp mini`

B. `ed mp mini ed`

C. `ed mp mini ed mini`

D. `ed mp mini ed mp mini`

E. Compilation fails.

F. `"ed mp mini"`, followed by an exception.

Pitanja s certifikata (2)

Given the proper `import` statements and:

```
23.  try {
24.    File file = new File("myFile.txt");
25.    PrintWriter pw = new PrintWriter(file);
26.    pw.println("line 1");
27.    pw.close();
28.    PrintWriter pw2 = new PrintWriter("myFile.txt");
29.    pw2.println("line 2");
30.    pw2.close();
31.  } catch (IOException e) { }
```

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

A. No file is created.

B. A file named `"myFile.txt"` is created.

C. Compilation fails due to an error on line 24.

D. Compilation fails due to an error on line 28.

E. `"myFile.txt"` contains only one line of data, `"line 1"`

F. `"myFile.txt"` contains only one line of data, `"line 2"`

G. `"myFile.txt"` contains two lines of data, `"line 1"` then `"line 2"`

Pitanja s certifikata (3)

Given:

```
import java.io.*;
public class ReadingFor {
 public static void main(String[] args) {
   String s;
   try {
     FileReader fr = new FileReader("myfile.txt");
     BufferedReader br = new BufferedReader(fr);
     while((s = br.readLine()) != null)
       System.out.println(s);
     br.flush();
   } catch (IOException e) { System.out.println("io error"); }
  }
}
```

And given that `myfile.txt` contains the following two lines of data:

```
ab
cd
```

What is the result?

A. `ab`

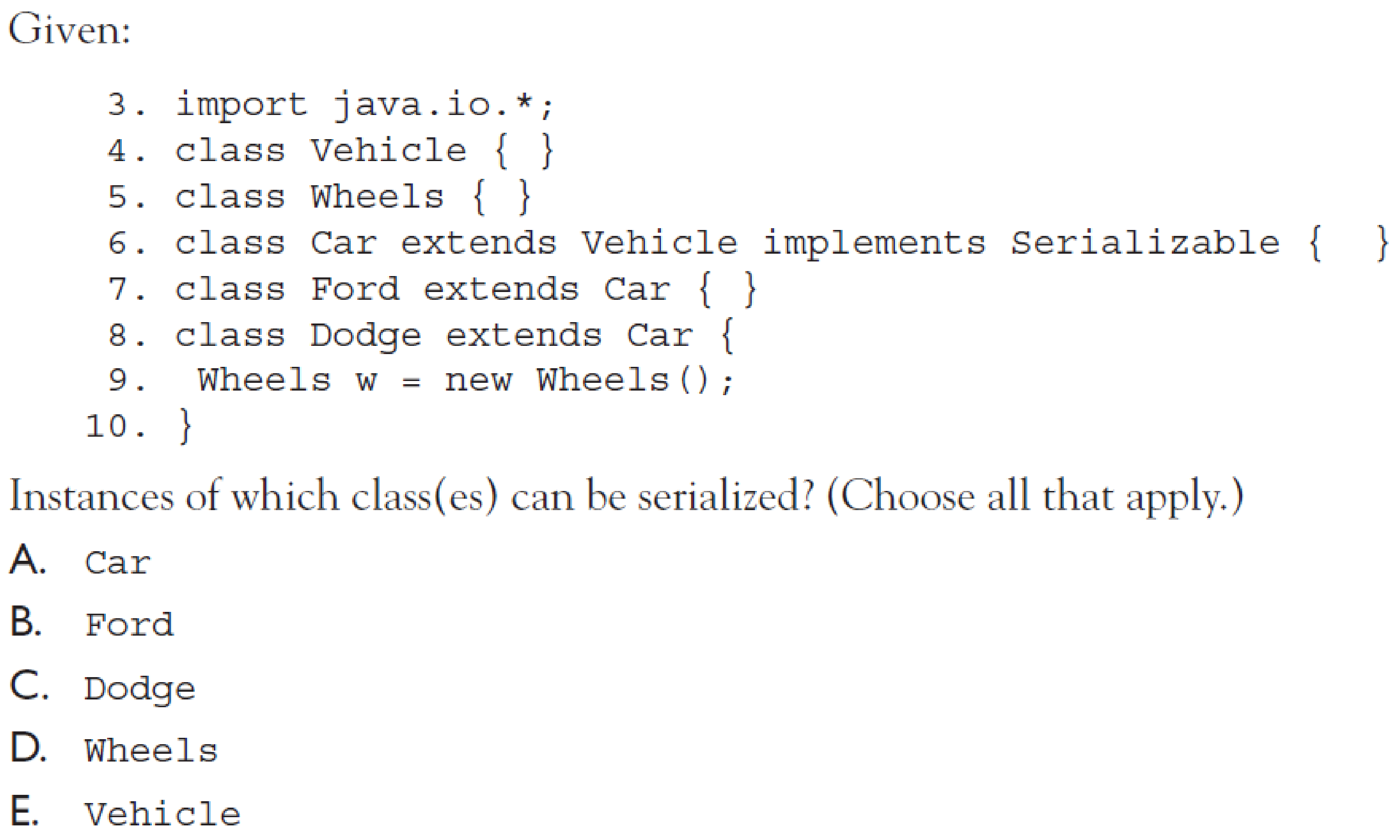
B. `abcd`

C.
```
ab
cd
```

D.
```
a
b
c
d
```

E. Compilation fails

Pitanja s certifikata (4)



Given:

```
import java.io.*;
class Vehicle { }
class Wheels { }
class Car extends Vehicle implements Serializable {   }
class Ford extends Car { }
class Dodge extends Car {
 Wheels w = new Wheels();
}
```

Instances of which class(es) can be serialized? (Choose all that apply.)

A. `Car`

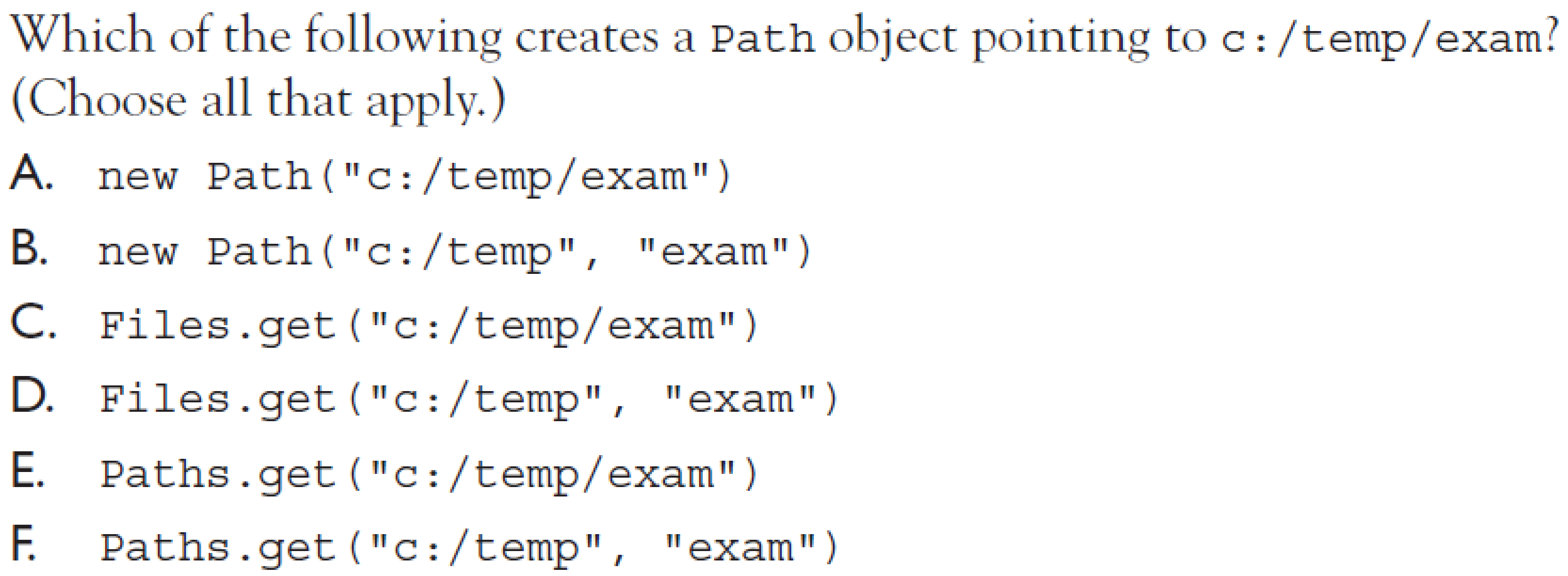
B. `Ford`

C. `Dodge`

D. `Wheels`

E. `Vehicle`

Pitanja s certifikata (5)



Which of the following creates a `Path` object pointing to `c:/temp/exam`? (Choose all that apply.)

A. `new Path("c:/temp/exam")`

B. `new Path("c:/temp", "exam")`

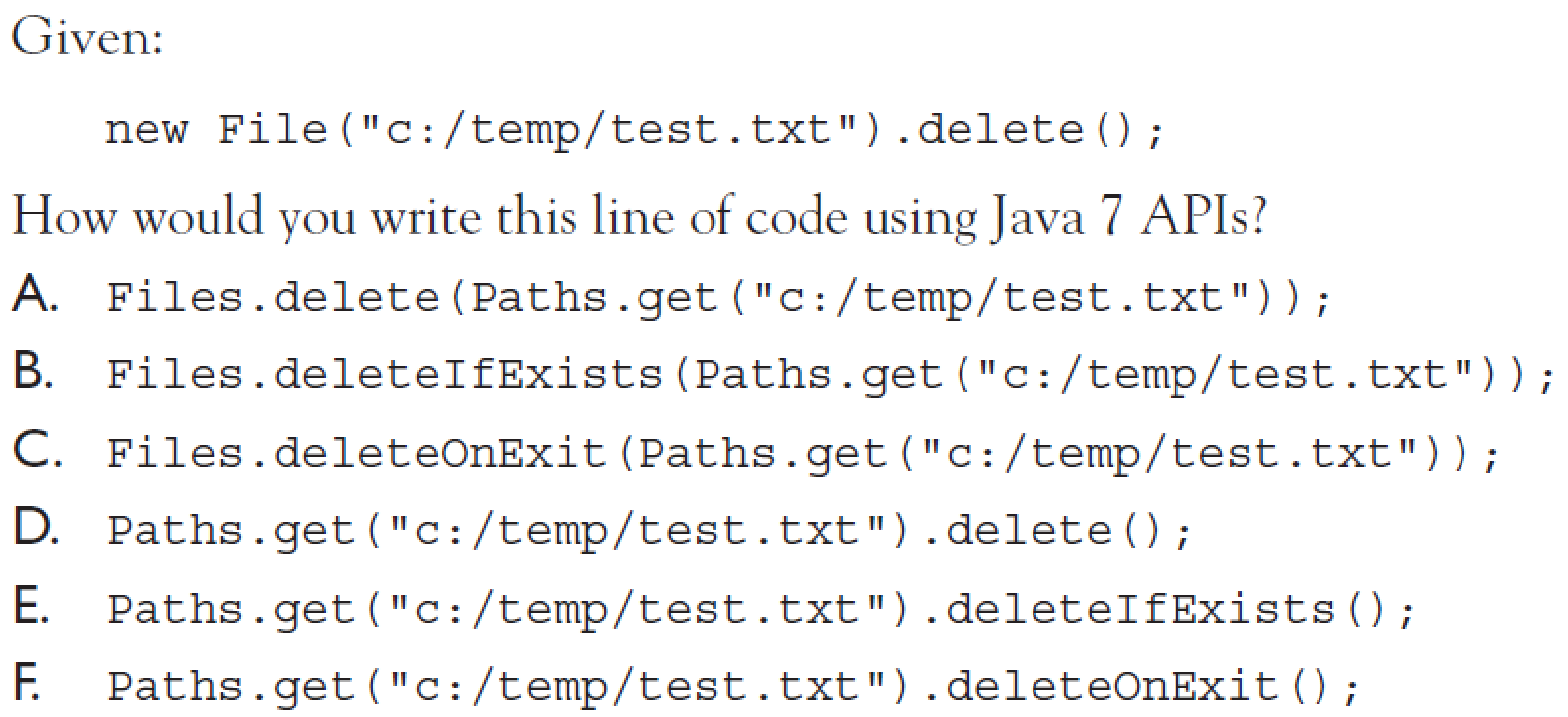
C. `Files.get("c:/temp/exam")`

D. `Files.get("c:/temp", "exam")`

E. `Paths.get("c:/temp/exam")`

F. `Paths.get("c:/temp", "exam")`

Pitanja s certifikata (6)



Given:

```
new File("c:/temp/test.txt").delete();
```

How would you write this line of code using Java 7 APIs?

A. `Files.delete(Paths.get("c:/temp/test.txt"));`

B. `Files.deleteIfExists(Paths.get("c:/temp/test.txt"));`

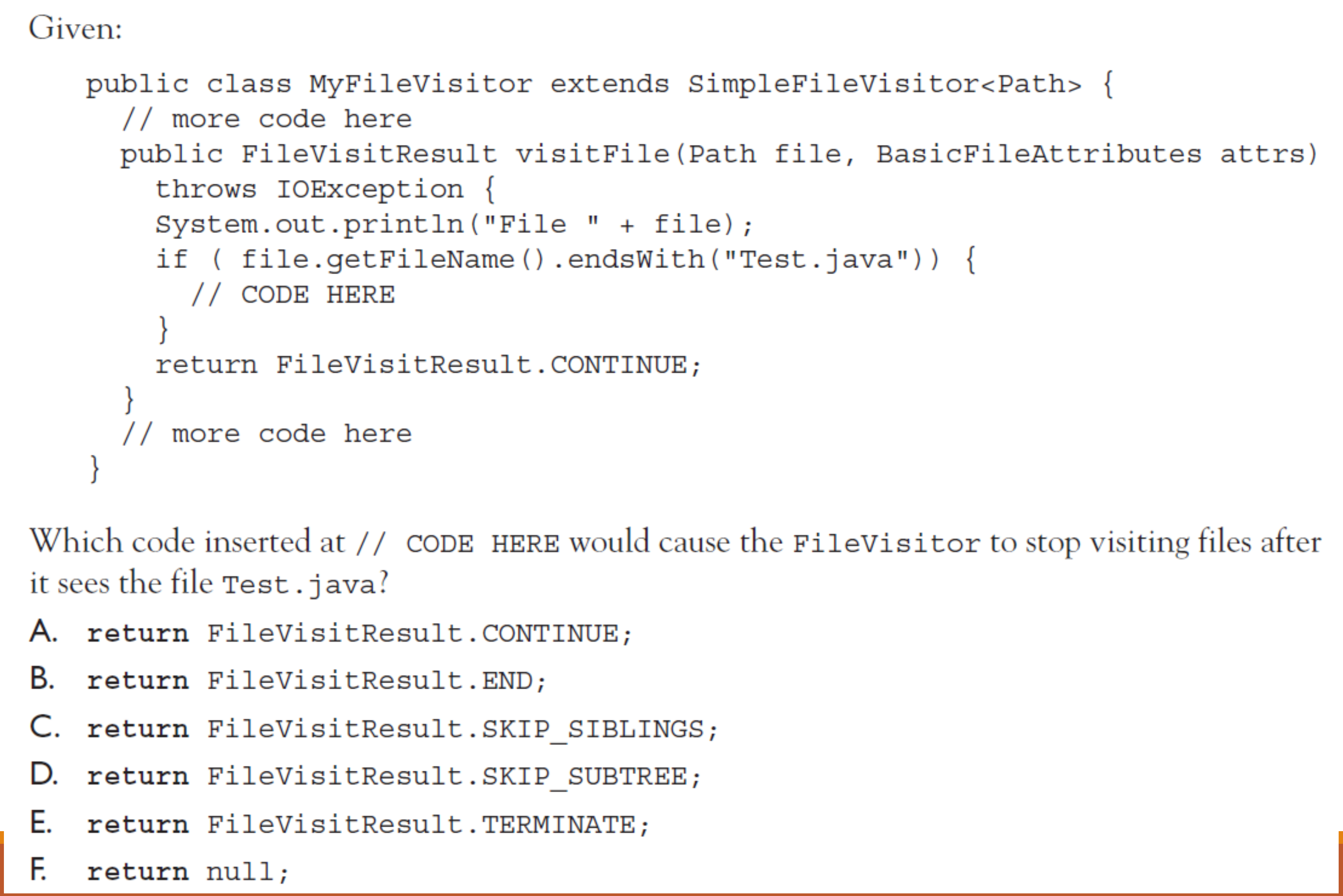
C. `Files.deleteOnExit(Paths.get("c:/temp/test.txt"));`

D. `Paths.get("c:/temp/test.txt").delete();`

E. `Paths.get("c:/temp/test.txt").deleteIfExists();`

F. `Paths.get("c:/temp/test.txt").deleteOnExit();`

Pitanja s certifikata (7)



Given:

```
public class MyFileVisitor extends SimpleFileVisitor<Path> {
  // more code here
  public FileVisitResult visitFile(Path file, BasicFileAttributes attrs)
    throws IOException {
    System.out.println("File " + file);
    if ( file.getFileName().endsWith("Test.java")) {
      // CODE HERE
    }
    return FileVisitResult.CONTINUE;
  }
  // more code here
}
```

Which code inserted at `// CODE HERE` would cause the `FileVisitor` to stop visiting files after it sees the file `Test.java`?

A. `return FileVisitResult.CONTINUE;`

B. `return FileVisitResult.END;`

C. `return FileVisitResult.SKIP_SIBLINGS;`

D. `return FileVisitResult.SKIP_SUBTREE;`

E. `return FileVisitResult.TERMINATE;`

F. `return null;`

Pitanja?