Exam 4: QUESTIONS

Question 1 of 4

Given the following,

```
1. interface Base {
2.   boolean m1 ();
3.   byte m2(short s);
4. }
```

Which code fragments will compile? (Choose 2 options)

A.
```
interface Base2 implements Base { }
```

B.
```
abstract class Class2 extends Base {
  public boolean m1() { return true; } }
```

C.
```
abstract class Class2 implements Base { }
```

D.
```
abstract class Class2 implements Base {
  public boolean m1() { return (true); } }
```

E.
```
class Class2 implements Base {
  boolean m1( ) { return false; }
  byte m2(short s) { return 42; } }
```

Question 2 of 4

Given:

```
import java.io.*;
class Master {
  String doFileStuff() throws FileNotFoundException { return "a"; }
}
class Slave extends Master {
  public static void main(String[] args) {
    String s = null;
    try { s = new Slave().doFileStuff();
    } catch ( Exception x) {
      s = "b"; }
    System.out.println(s);
  }
  // insert code here
}
```

Which, inserted independently at **// insert code here**, will compile, and produce the output b? (Choose 4 options)

A. `String doFileStuff() { return "b"; }`
B. `String doFileStuff() throws IOException ( return "b"; }`
C. `String doFileStuff(int x) throws IOException ( return "b"; }`
D. `String doFileStuff() throws FileNotFoundException { return "b"; }`
E. `String doFileStuff() throws NumberFormatException { return "b"; }`
F. `String doFileStuff() throws NumberFormatException, FileNotFoundException { return "b"; }`

Question 3 of 4

Given:

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

Which could be used to create an appropriate catch block? (Choose two options.)

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

Question 4 of 4

Which will compile and run without exception? (Choose 4 options)

A. `System.out.format("%b", 123);`
B. `System.out.format("%c", "x");`
C. `System.out.printf("%d", 123);`
D. `System.out.printf("%f", 123);`
E. `System.out.printf("%d", 123.45);`
F. `System. out. printf ("%f", 123.45);`
G. `System.out.format("%s", "123");`

```
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.

SOLUTIONS

Question 1 of 4

Given the following,

```
1. interface Base {
2.   boolean m1 ();
3.   byte m2(short s);
4. }
```

Which code fragments will compile? (Choose 2 options.)

A.
```
interface Base2 implements Base { }
```

B.
```
abstract class Class2 extends Base {
  public boolean m1() { return true; } }
```

C.
```
abstract class Class2 implements Base { }
```

D.
```
abstract class Class2 implements Base {
  public boolean m1() { return (true); } }
```

E.
```
class Class2 implements Base {
  boolean m1( ) { return false; }
  byte m2(short s) { return 42; } }
```

1. ☑ **C** and **D** are correct, **C** is correct because an abstract class doesn't have to implement any or all of its interface's methods. **D** is correct because the method is correctly implemented.
   ☒ **A** is incorrect because interfaces don't implement anything, **B** is incorrect because classes don't extend interfaces. **E** is incorrect because interface methods are implicitly public, so the methods being implemented must be public.

Given:

```
import java.io.*;
class Master {
  String doFileStuff() throws FileNotFoundException { return "a"; }
}
class Slave extends Master {
  public static void main(String[] args) {
    String s = null;
    try { s = new Slave().doFileStuff();
    } catch ( Exception x) {
      s = "b"; }
    System.out.println(s);
  }
  // insert code here
}
```

Which, inserted independently at **// insert code here**, will compile, and produce the output b? (Choose 4 options)

A. String doFileStuff() { return "b"; }
B. String doFileStuff() throws IOException (return "b"; }
C. String doFileStuff(int. x) throws IOException (return "b"; }
D. String doFileStuff() throws FileNotFoundException { return "b"; }
E. String doFileStuff() throws NumberFormatException { return "b"; }
F. String doFileStuff() throws NumberFormatException, FileNotFoundException { return "b"; }

☑ **A, D, E,** and **F** are correct. It's okay for an overriding method to throw the same exceptions, narrower exceptions, or no exceptions. And it's okay for the overriding method to throw any runtime exceptions.
☒ **B** is incorrect, because the overriding method is trying to throw a broader exception. **C** is incorrect. This method doesn't override, so the output is a.

Question 3 of 4

Given:

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

Which could be used to create an appropriate catch block?

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

☑ **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.

Question 4 of 4

Which will compile and run without exception?

```
A. System.out.format("%b", 123);
B. System.out.format("%c", "x");
C. System.out.printf("%d", 123);
D. System.out.printf("%f", 123);
E. System.out.printf("%d", 123.45);
F. System out. printf ("%f", 123.45);
G. System.out.format("%s", "123");
```

☑ **A, C, F**, and **G** are correct. The %b (boolean) conversion character returns true for any non-null or non-boolean argument.
☒ B is incorrect, the %c (character) conversion character expects a character, not a String, ,D is incorrect, the %f (floating point) conversion character won't automatically promote an integer type. E is incorrect, the %d (integral) conversion character won't take a floating point number. (Note; The format() and printf() methods behave identically.)