

1. Which is required for refactoring to take place?

- a. Use cases
- b. Known requirements
- c. Functioning Code
- d. Test Cases
- e. State Diagrams

2. class Stadium {

```
    ...
    double getTicketPrice() {
        double returnValue = 0;
        switch (getConcert().getEventClass()) {
            case Concert.PREMIUM:
                returnValue = getAmount() * 2;
                break;
            case Concert.NORMAL:
                returnValue = getAmount();
                break;
            case Concert.BASIC:
                returnValue = getAmount() * 0.5;
                break;
        }
        return returnValue;
    }
}
```

How do you use polymorphism during code refactoring for the sample code above?

- a. Move the calculation of the ticket price to distinct subclasses of the Concert class.
- b. Inline the getAmount() code into each case statement.
- c. Use a series of "if-else" clauses instead of a switch statement.
- d. Pass the Concert enum to another method of Stadium to calculate the ticket price.
- e. Create a static Map where the key is the Concert type and the value is the associated price factor, and get that factor at runtime.

3. Sample Code

```
String strCase1 = someTest.getSomeProperties1();
strCase1 = doSomething(strCase1);
String strCase2 = someTest.getSomeProperties2();
strCase2 = doSomething(strCase2);
String strResult = strCase1 + strCase2;
someTest.setSomeProperties(strResult);
```

How do you refactor the sample code above to function the same?

- a. someProperties.setSring(

doSomething(someCase1.getSomeProperties1()) +

doSomething(someCase2.getSomeProperties2()));
- b. someResult.setSomething(doCase1(someProperties.getStr())

+ doCase2(someProperties.getStr()));
- c. someTest.setProperties(

doTest(someResult.getSomeProperties1()) +

doTest(someResult.getSomeProperties2()));
- d. someTest.setSomeProperties(

doSomething(someTest.getSomeProperties1()) +

doSomething(someTest.getSomeProperties2()));
- e. someCase.setSomething(

doSomething(someCase.getSomeResult1())

+doSomething(someCase.getSomeResult2()));

4. scenario

A program has numerous Service and DAO classes that have the same 30-line exception code block set which repeats throughout the code.

Based on the scenario above, how do you refactor the code?

- a. Throw the exception handling into a loop, and run it each time an error occurs.
- b. Insert a try-catch-finally block of code where the exception code block occurs.
- c. Go through and rewrite the code so it has more documentation and is easier to read.
- d. Extract the exception handling into a method, and then call that method as necessary.
- e. Change the observable behavior of the code so it operates more efficiently.

5. Line 1 package com.sample.TestExamples;
 Line 2 import java.io.*;
 Line 3 public class DocumentationSample
 Line 4 implements Serializable {
 Line 5 // *** Additional Code ***
 Line 6 }

Based on the sample code above, to provide a documentation comment that describes the purpose of the class

DocumentationSample, you insert:

- /**java.awt.../ into the code above the class
- @see java.awt... into the code below the class
- @see java.awt... into the code above the class
- /** java.awt... into the code below the class
- < ** java.awt... into the code below the class

6. Sample Code

```
public void aMethod(String arg) {
    ...
}
```

In the sample code above, which Method comment do you use to document the parameter in a single word Description in javadoc?

- @parameter Description
- @param Description
- @parameter String arg Description
- @parameter method arg Description
- @param arg Description

7. When using the javadoc utility, which statement do you use to format javadoc-style comments?

- <!-- comment -->
- @doc{{ comment }}
- /** comment */
- // comment //
- /* comment */

8. Line 1 public @interface Example {
 Line 2 String newValue();
 Line 3 String oldValue();
 Line 4 }

Which code snippet do you insert above Line 1 in the sample code above to complete the creation of a custom annotation?

- a. object.getClass().getAnnotations()
 = m.Annotations
- b. annotation = m.getAnnotation
 create new
- c. @Target(ElementType.***)
 @Retention(RetentionPolicy.***)
- d. java.lang.Annotation
 setting.class.getAnnotation
- e. AnnotationProcessor
 get results

9. public class Test {
 public static void main(String args[]) {
 int i = args.length;
 switch (i) {
 case 0: System.out.println("Zero");
 case 1: System.out.println("One");
 case 2: System.out.println("Two");
 case 3: System.out.println("Three");
 default: System.out.println("Default");
 }
 }
 }

Where do you add @SuppressWarnings("fallthrough") annotation to the sample code above for the "javac -Xlint:fallthrough Test.java" command to not generate any warnings?

- a. Before each case clause that falls through
- b. Within the javadoc for the class
- c. Within the class import declaration
- d. Immediately above the switch statement
- e. Above the main method declaration

10. Code

```
public class Sample {
    String name;
    private Sample(String name) {
        this.name = name;
    }
    @Override
    public int hashCode() {
        return name.hashCode();
    }
    @deprecated
    public static String getCompanyName() {
        return "Example";
    }
    public String getName() {
        return name;
    }
}
```

Why is there a compilation error in the sample code above?

- a. The annotation for deprecated methods is @Deprecated, not @deprecated.
- b. The class does not have public constructors.
- c. The annotation for overriding methods is @override, not @Override.
- d. The hash code method is spelled hashCode, not hashcode.
- e. The class does not have a public no-argument constructor.

11. What do you call the concept of renaming classes and variables to protect your java code from being reverse engineered and easily read by someone else?

- a. Commenting
- b. Obfuscating
- c. Encrypting
- d. Decompiling
- e. Serializing

12. Which class must be extended by all notification events for JavaBean components?

- a. EventListener
- b. EventObject
- c. Object
- d. Serializable
- e. AWTEvent

13. Sample Code

```
public class TestThis {
    private int int_value;
    public static void main(String[] args){
        TestThis testThis = new TestThis();
        testThis.setInt_value(1);
        int integer = testThis.getInt_value();
    }
    public TestThis(){}
    public int getInt_value(){
        return int_value;
    }
    public void setInt_value(int value){
        int_value = value;
    }
}
```

Based on the sample code above, how do you start the debugger and then set a breakpoint in the code?

- a. run TestThis
 >halt at <class id>:<line>
- b. jdb TestThis
 >stop at <class id>:<line>
- c. begin TestThis
 >pause at <class id>:<line>
- d. start TestThis
 >break at <class id>:<line>
- e. jvm TestThis
 >debug at <class id>:<line>

14. Which command line do you use to load the JDWP agent for debugging?

- a. run=-Xjdwp:<sub-options>
- b. -runjdwp=<name1>[=<value1>],<name2>[=<value2>]...
- c. -agentlib:jdwp=<sub-options>
- d. -attach host:port
- e. -agent:jdwp=<sub-options>

15. Which command line for jdp do you use to remotely debug a Java program?
- a. -agentlib:jdwp
 - b. -ea.debug:run
 - c. -help:jdwp
 - d. -JWMdebug
 - e. -Xdebug -Xrunjdwp
16. Which command line do you use in jdb debugger to start debugging the TestThis class?
- a. jdb > TestThis.main()
 - b. jdb > run TestThis
 - c. jdb -run TestThis
 - d. jdb > start TestThis
 - e. jdb TestThis
17. When creating a JavaBean component, how do you make its properties available to users?
- a. Call a custom BeanInfo string that declares the properties public.
 - b. Declare the Bean's fields public.
 - c. Include public accessor methods of the form getVar() and setVar(val).
 - d. Declare an array of type java.lang.reflect.Field which contains the property names that should be available.
 - e. Implement support for ActionListener listeners for the fields.
18. Which class do you use to manage a list of listeners for a JavaBean component?
- a. ControllerEventListener
 - b. ContextList
 - c. ChangeListener
 - d. AWTEventListenerProxy
 - e. CopyOnWriteArrayList

19.

```
public void makeValueExample(Object o) {
    // o is an object of an unknown class that follows
    // JavaBean Naming Conventions

    // Set the object's Value property to "Example".
    try{
        //insert code here
    }catch (Exception e){
        System.out.println("Couldn't set the value");
    }
}
```

In the sample code above, which line of code do you insert in place of "insert code here" to set the Value property to Example?

- a. `Class.getMethod(Class.forName(o), new String("value")).invoke(o, new String[]{"Example"})`;
- b. `o.getClass().setProperty("Value","Example")`;
- c. `o.getClass().getMethod("setValue").invoke(o, "Example")`;
- d. `new PropertyDescriptor("value",o.getClass()).getWriteMethod().invoke(o, "Example")`;
- e. `Bean.getAllPropertyMethods(o).invoke("setValue(java.lang.String)","Example")`;

20. In the Sample code below, on which line you must place a documentation comment that describes the purpose of the class DocumentationSample?

```
//Line A
package com.brainbench.TestExamples;
//Line B
import java.io.*;
//Line C
public class DocumentationSample
//Line D
    implements Serializable{
//Line E
    //Some code
}
```

- a. LINE A
- b. LINE B
- c. LINE C
- d. LINE D
- e. LINE E