1. (1 point) Which of the following is true of a well–executed refactoring?
    A. It relies on inheritance and polymorphism
    B. It has intermediate steps where the code is functional
    C. It uses a refactoring IDE like Eclipse
    D. It creates a new class (or maybe several)
    E. When the refactoring is finished, you usually need to add several unit tests to exercise the new functionality

2. (1 point) Which of the follow describes a Refused Bequest?
    A. NetworkedFile, a subclass of GameDataFile, that when you call saveToDisk() actually writes the file to a cloud storage across the network
    B. AutosaveRecord that has a variable NetworkHanlder that is usually null
    C. Manager, a subclass of employee, that returns -1 when the getEmployeeId() method is called because managers don't have employee ids
    D. LoginCommand, which duplicates many methods of NetworkCommand but is not Network-Commands' subclass
    E. CompositeWindow is an abstract class designed to be a superclass that has no subclasses

3. (1 point) What smell does this source code suggest?

```
class Student
 {
  private String name;
  private int gradYear;
  private int studentID;

  public String getName() {...}
  public void setName(String name) {...}
  public int getGradYear() {...}
  public void setGradYear(int year) {...}
  public int getStudentID() {...}
  public void setStudentID(int id) {...}
}
```

    A. Data Class
    B. Short Class
    C. Refused Bequest
    D. Data Clumps
    E. Actually, this code is fine

4. (1 point) You and your friend are looking at the function signature below. Your fiends suggests that "this might be an instance of Primitive Obsession". What might your friend be proposing?

```
public int getCustomerIdForReferral(int customerIdOfReferrer,
                                    double referrerPercent,
                                    String url,
                                    Product p)
```

    A. That this function has a large number of parameters, many of them Java primitives, and that the number of parameters should be reduced
    B. That referrerPercent, url, and p could be combined into a single object

C. That this method would make more sense if it was a method on the Product class

D. That the function would be improved if it used existing Java classes Integer and Double rather than int and double

E. That a new CustomerId object might be created, rather than using ints for customer ids

5. (1 point) Which of these would be an example of divergent change?

A. A FormatParser class that needs to be subclassed in one way when you add a new add format, and another way when you add a new output type

B. A Sprite class in a video game and that you subclass every time you need a new kind of sprite and implement 3 different abstract methods

C. A HTTPProtocol class that has one gigantic method that every new feature needs to add to

D. A web system where you have to both update the C++ backend code as well as the Perl webpage code

E. A system where everytime you add a new DataElement class, you also need to add a new DataElementRenderer class

6. (1 point) Under what circumstances might you want to take two existing classes and give them a common superclass?

A. Both classes have similar methods and you can remove duplication by moving them to the superclass

B. One class has several Temporary Variables that can be Pulled Up into the superclass

C. You need to use the superclass as a Middle Man for the clients of both of the classes

D. Both classes are part of Parallel Inheritance Hierarchies and want to remove the implicit duplcation by giving both hierarchies a shared interface

E. One class is a Large Class and moving methods into the superclass will make it smaller

7. (1 point) You're reading some code and you come across the class below. What conclusions do you draw?

```
abstract class HTMLTableWriter {
  public void outputTableHeader() {...}
  public void outputTableFooter() {...}
  public abstract void outputTableHeadings();
  public abstract void outputTableContents();

  public void outputTable() {
    outputTableHeader();
    outputTableHeadings();
    outputTableContents();
    outputTableFooter();
  }
}
```

A. This superclass is an example of Speculative Generality and it's methods should be Pushed Down into it's subclasses

B. The method outputTable is probably overridden in HTMLTableWriter's subclasses. Because HTMLTableWriter is abstract the implementation here can't be called.

C. This class could be improved by moving outputTableHeader and outputTableFooter into a newly created class

D. The outputTable method is a Template Method

E. The outputTable method isn't really accomplishing anything — it might be worthwhile to use Inline Method on outputTableHeader and outputTableFooter

8. (1 point) You come across the following code. What refactoring would most improve it?

```
public void updateName(DataRecord newFile) {
  string result = null;
  while(newFile.hasNext()) {
    DataRecordEntry e = newFile.getNext();
    if(e.key().equals("name")) { result = e.value(); }
  }
  if(result == null) throw new RuntimeException("name not found");
  name = result;
}

public void updateDescription(DataRecord newFile) {
  string result = null;
  for(DataRecordEntry i = newFile.getNext(); newFile.hasNext(); i = newFile.getNext())
  {
    if(i.key().equals("description")) { result = i.value(); }
  }
  if(result == null) throw new RuntimeException("description not found");
  description = result;
}
```

A. A single utility method should be extracted and called from both functions, eliminating the duplication

B. The variables e and i should be renamed to be more explanatory

C. A local variable should be introduced to explain the method's purpose more clearly

D. The functions should be changed to return an error code rather than throwing an exception when problems are found

E. i.key().equals(...) is a Message Chain and should be removed

9. (1 point) Object Oriented programmers often say that switch statements are bad. Why?

A. Case statements encourage writing long methods

B. In languages like Java, strings cannot be used in case statements so they require you to use hard coded constants

C. Case statements are often vary behavior based on types, which can be replaced by polymorphism

D. Case statements introduce a strong performance overhead in OO languages because they can't be optimized the same way they can be in procedural languages like C

E. Case statements often have subtle bugs which more straightforward if statements don't

10. (1 point) In an error reporting system, you notice a lot of the classes tend to have the same set of 3 instance variables: url, customerId, and timestamp. What might this suggest?

A. That these three variables often occur together, and should be replaced with a single identifier that links to a global map

B. That these three variables are a potential source of memory overhead, and you should profile your code to check

C. That these three objects might be extracted out into a single class

D. That your classes are likely repeating data and should be combined into one class

E. That there should be a common superclass of all the classes in the system, and these three fields should be protected members

11. (20 points) Speculative generality is one of the more complex code smells. Obviously we want to make our code flexible — indeed courses like CSSE 374 are about teaching us how to do that. If we can see an opportunity to implement a particular design pattern, for example, why not implement it?
    (a) (5 points) Why does Fowler think of Speculative Generality as a bad thing?
    (b) (5 points) Write some sample code or UML that exhibit an example of speculative generality. Explain briefly what about your example shows speculative generality.
    (c) (5 points) Normally there'd be some more parts, but this is just a sample