Queens College, CUNY
Department of Computer Science

CS 212 – Object-Oriented Programming in Java – Practice Exam 2

CS 212 Exam 2 Study Guide

Topics for Exam 2:

Linked Lists

- define a list node
- define a singly-linked list with head node and counter
- be able to traverse the list from beginning to end and, for example, find a given node, print the data in a given node.
- append a new node to the end of the list.
- insert a new node after a given node in the list

Interfaces

- understand the difference between a *Class* and an *Interface*
- understand that classes are *extended* and interfaces are *implemented*

Inheritance

- understand what methods and data items are inherited from a super class when they are declared public, private and protected.
- understand the use of *super()* , the default use of *super()* and how to work with multiple *super()* methods (with different signatures).
- understand the use of the *this* reference to use different constructors in the same class.
- understand the use of overridden methods (polymorphism) in a class hierarchy (for example, for a given object, which *toString* method will be called.

Abstract Classes and Methods

- understand what the implications are when a class or methods in a class are declared abstract (that is, an abstract class cannot be instantiated and must be extended to the point that no methods remain abstract.)

Event-Driven Programming

- Creation of a GUI (JFrame) with JMenuBar, JMenu, and JMenuItem.
- Registration of JMenu items with an even handler (ActionListener)
- Creation of an ActionListener with an appropriate actionPerformed method.
- Determine which menu item generated the event

Question 1 Linked lists.

Assuming that a singly-linked list with head node containing Social Security numbers (as in the Lecture on Dynamic Structures) been created correctly, write a method for the SSNList class will print each SSN in the list to the system console.

```
Public void printNodes (SSNList myList) {
        SSNListNode p;
        p=SSNList.first.next;
        while (p != null) {
            System.out.println(p.data);
            p=p.next;
        }
}
```

Question 2- Interfaces.

a.

```
public interface Exam2 {
      public int getAnswer(int i, int j);
      public String getString(String[] s, int i);
}
public Class Exam2class implements Exam2 {

}
```

What methods must be declared in Class Exam2Class?

Both getAnswer and getString

b.

```
public Interface Exam2 {
      public int getAnswer(int i, int j);
      public String getString(String[] s, int i);
}

public Class Exam2Super {
      public int getAnswer(int p, int q) {
         return p+q;
      }
}

public Class Exam2class extends Exam2Super implements Exam2 {

}
```

What methods must be declared in Class Exam2Class?

getString must be declared, getAnswer may be overridden.

Question 3 - Inheritance.

Examine the code below and answer the following questions.

a. Provide a *get method* to be inserted after line 13 for the value of *species*.

```
public String getSpecies() {
    return species;
}
```

b. Provide a *set method* for the instance variable *population* which does the same error checking as the constructor.

```
public setPopulation(int p) {
    if (p<0)
        throw new InvalidAnimalException("Population cannot be negative");
    population =p;
}
```

c. Write a class *FlyingAnimalException* which is an extension of *IllegalArgumentException.*

```
public class FlyingAnimalException extends IllegalArgumentException {
    public FlyingAnimalException(String message) {
        super (message);
    }
}
```

d. The instance variable *wingBeatsPerSecond* in the abstract class *FlyingAnimal* must be greater than zero. Complete the *if* statement on line 20 so that a *FlyingAnimalException* is thrown with the message "Wing beats must be greater than zero." if it is not.

```
if(w < 1) throw new FlyingAnimalException("Wing beats must be greater than zero");
```

e. Provide the proper arguments for the *super* call in the single-argument constructor on line 32 so that the default values from the interface are used for the remaining parameters.

```
super(species, Flapping.DEFAULT_POPULATION,Flapping.DEFAULT_WINGBEATS);
```

```
public interface Flapping {
   public static final int DEFAULT_WING_BEATS = 30;
   public static final int DEFAULT_POPULATION = 10000;
   public int wingBeats ();
   public boolean flaps ();
}
```

```
 1  public abstract class Animal {
 2     String species;
 3     int population;
```

```
4
5     public Animal (String s, int p) {
6        if (s == null || s=="")
7           throw new InvalidAnimalException("Species must be given.");
8        if (p<0)
9           throw new InvalidAnimalException("Population cannot be negative");
10       species = s;
11       population =p;
         System.out.println("New animal, species is "+species+",
12              population "+population);
13    }
14 }
```

```
15
16 public abstract class FlyingAnimal extends Animal implements Flapping {
17    protected int wingBeatsPerSecond;
18    public FlyingAnimal (String name, int population, int w) {
19       super(name,population);
20       if
21       else
22          wingBeatsPerSecond = w;
23    }
24    public String toString() {
25       return this.getClass().getName();
26    }
27 }
```

```
28
29 public class Bird extends FlyingAnimal  {
30    private boolean flapping = true;
31    public Bird (String species) {
32       super (                                                    );
33    }
34    public Bird (String species, int population, int wingBeats) {
35       super(species, population, wingBeats);
         System.out.println(this.toString()+" created, Species is "+
36                  species+" wing beats = "+wingBeats);
37    }
38    public int wingBeats() {
39       return wingBeatsPerSecond;
40    }
41    public boolean flaps (){
42       return flapping;
43    }
44 }
```

```
45
46 public class Bat extends FlyingAnimal  {
47    private boolean flapping = false;
48    public Bat (String species) {
49       super (species, Flapping.DEFAULT_WING_BEATS,Flapping.DEFAULT_POPULATION);
50    }
51    public Bat (String species, int population, int wingBeats) {
52       super(species, population, wingBeats);
         System.out.println(this.getClass().getName()+" created, Species is "+
53                  species+" wing beats = "+wingBeats);
54    }
55    public int wingBeats() {
56       return wingBeatsPerSecond;
57    }
58    public boolean flaps (){
59       return flapping;
60    }
```

Question 4. (Continuing with the set of classes above)

```
public class Main {
   public static void main(String[] args) {
      Bat igor=null;
      Bird robin;
      FlyingAnimal finch;
      Statement 1;
   }
}
```

What will be the output of the application above for each given value of ***Statement1.*** If no value is given, assume the statement is blank. If either statement will cause a *compiler* error, explain why.

a. ***Statement1*** is `igor = new Bat("Vampire",1200,30);`

```
New animal, species is Vampire, population 1200
Bat created, Species is Vampire wing beats = 30
```

b. ***Statement1*** is `igor = new Bat(null,1200,30,false);`

```
Exception in thread "main" InvalidAnimalException: Species must be given.
```

c. ***Statement1*** is `robin = new Bird ("Robin",3000,50);`

```
New animal, species is Robin, population 3000
Bird created, Species is Robin wing beats = 50
```

d. ***Statement1*** is `robin = new Bird ("Robin");`

```
New animal, species is Robin, population 30
```

e. ***Statement1*** is `igor = new Bat ("",20,25);`

```
Exception in thread "main" InvalidAnimalException: Species must be given.
```

f. ***Statement1*** is `finch = new FlyingAnimal ("Finch",5000,45);`

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
   Cannot instantiate the type FlyingAnimal
```

.

Question 5.

The following code is meant to create a GUI (JFrame) with one menu called “Exam 2” which has two menu items, “Pass” and “Fail”. Write the proper code for each of the missing pieces in the box to the right.

```
import java.awt.*;
import javax.swing.*;

public class MainWindow extends [ 1 ]      {

   private JFrame mainWindow;
   private [ 2 ]     mainMenuBar;
   private Menu [ 3 ];
   private [ 4 ]     mPass, mFail, [ 5 ]   ;
   private [ 6 ] exam2ML;

   public static void main (String args[]){
      MainWindow m = new MainWindow("Exam 2");
   }

   public MainWindow (String title) {
      mainWindow = new JFrame(title);
      mPass = new [ 7 ] ("Pass");
      mFail = new       ("Fail");
      mQuit = new       ("Quit");
      exam2 = new Menu ("Exam 2", /*tearoff =*/ false);
      exam2.add(mPass);
      exam2.add(mFail);
      exam2.addSeparator();
      exam2.add(mQuit);
      mainMenuBar = new MenuBar();
      mainMenuBar.add( [ 8 ] );
      mainWindow.setMenuBar(mainMenuBar);
      exam2ML = new [ 9 ];
      mPass. [ 10 ] (exam2ML);
      mFail.         (exam2ML);
      mQuit.         (exam2ML);

      mainWindow.setDefaultCloseOperation(EXIT_ON_CLOSE);
      mainWindow.setSize(400,400);
      mainWindow.setLocation(100,100);
      mainWindow.setVisible(true);

   } // constructor
} // class
```

Answers:

1._____JFrame_____

2.___JMenuBar_____

3.___exam2_____

4.___JMenuItem_____

5.___mQuit_____

6.___Exam2Listener_____

7.____JMenuItem_____

8.____exam2_____

9.____Exam2Listener(this)

10.__addActionListener___

Question 6.

The following code is a listener for the GUI in the question above. If the user chooses the menu item “Pass”, as message saying “Hooray! Your passed!” should be printed to the console. If the user chooses “Fail”, a message saying “Too bad! You failed.” should be printed to the console. If the user chooses “Exit”, the program should terminate. Write the proper code for each of the missing pieces in the box to the right.

```
import java.awt.*;
import java.awt.event.*;
public class Exam2Listener implements [ 1 ] {

   private [ 2 ] mainFrame;
   public Exam2Listener(MainWindow f) {
      mainFrame = f;
   }  // constructor

   public void [ 3 ](ActionEvent e) {
      String chosenItem = ((MenuItem) e.getSource()).getLabel();
      if (chosenItem.equals( [ 4 ] )) {
          System.out.println("Hooray! You passed!");
      } // end if mInputFile
      else if (chosenItem.equals("Fail")) {
          System.out.println( [ 5 ] );
      } // end if mInputFile
      else if ( chosenItem.equals("Quit")) {
          System.exit(0);
      }  //if
    }  //
} // class Exam2Listener
```

Answers:

1._ActionListener_____

2.__MainWindow_____

3.__actionPerformed_____

4.__"Pass"_____

5.___"Too Bad! You failed."