

CSE 120

Midterm II — March 21, 2007

Name: _____

Student ID: _____
(from your PennCard)

Email _____ @seas.upenn.edu

Lab:(circle one)

201	202	203	204
W 1:00	W 2:30	R 12:00	R 1:30

- This exam text and attached code, the *Java Backpack Reference Guide*, and your brain are the only information sources you can use during this exam.
- You have 50 minutes to answer all of the questions. The entire exam is worth 50 points. The point value of each question is given to help you plan your time.
- Write your answers on the exam pages. The back side of each page may be used as a scratch pad.
- The answers to the questions require only basic Java language facilities and the code and APIs given in the appendix. No other classes or packages are needed.
- Questions during the exam should be about the wording of the exam only. If you have a question, raise your hand and we'll come to you. (This is less disruptive for others than if you come to us.)
- **Partial credit will be given where appropriate.**
- DON'T PANIC!. If you find a question that you cannot solve right away, consider moving on and returning to it after you finish the other questions.

	Score	Max
1		16
2		12
3		12
4		10
Total		50

1. Consider a simple implementation of a generic linked list. The beginning of this implementation is shown below.

```
import java.util.*;

public class SimpleLinkedList<E> {
    private int size;
    private Node first;

    private class Node {
        E element;
        Node next;
        Node (E element, Node next) {
            this.element = element;
            this.next = next;
        }
    }

    // create an empty list
    public SimpleLinkedList() {
        first = null;
        size = 0;
    }

    // return the number of elements in the list
    public int size() { return size; }

    // return an iterator for the list elements
    public Iterator<E> iterator() { return new LinkedListIterator(); }
```

- (a) (6 points) Complete the definition of the `addFirst` method that inserts an element at the *beginning* of the list.

```
    // add an element to the front of the list
    public void addFirst(E element) {
        Node newNode = new Node(element, first);
        first = newNode;
        size++;
    }
```

- (b) (10 points) Complete the definition of the inner class `LinkedListIterator`, implementing the `Iterator<E>` interface (shown in the Appendix). Your implementation of the `remove` method should just throw an `UnsupportedOperationException`, instead of removing any elements from the list. You may assume that the list will not be modified during iteration, and you do not need to detect or report any such modifications.

Note that the constructor of this class is called in the `iterator` method above. `LinkedListIterator` should include a constructor that takes no arguments. (Remember that because `LinkedListIterator` is an inner class, it has access to the private variables of `SimpleLinkedList`).

```
private class LinkedListIterator implements Iterator<E> {
    Node cursor;
    LinkedListIterator() {
        cursor = first;
    }
    public boolean hasNext() {
        return (cursor != null);
    }
    public E next(){
        if (cursor != null) {
            E e = cursor.element;
            cursor = cursor.next;
            return e;
        } else {
            throw new NoSuchElementException();
        }
    }
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Grading guidelines:

For part (a) the 6 points were divided as follows:

- 3 Creating a new node correctly*
- 2 Updating first node*
- 1 Modifying size*

For part (b) the 10 points were divided as follows:

- 1 instance variable/variables*
- 2 constructor*
- 2 hasNext*
- 3 next*
- 2 remove*

2. (12 points) The next problem concerns the implementation of the method `askAge`.

The static method, `askAge` in the class `Age`, prompts the user for his or her age and returns it. If the user does not enter a number between the range of 0 to 120 (inclusive), the method prompts the user for another response. (If the user types a number, and then some whitespace and then some additional text, that text should just be ignored.) For example, a DrJava interaction might look like:

```
> Age.askAge(System.in)
Please input your age:
  1ab3
Age should be a number between 0 and 120. Please try again.
  2000
Age should be a number between 0 and 120. Please try again.
  2 years
2
>
```

In the implementation of `askAge`, all input should be from the parameter `input` of type `InputStream`, and all output should be to `System.out`, using the method `println`. Input should be processed using an instance of the class `java.util.Scanner` created in the following manner:

```
Scanner scanner = new Scanner(input);
```

To read input, `askAge` should use the method `next` from the `java.util.Scanner` class. The method `askAge` may also use the static method `parseInt` from the `java.lang.Integer` class. More information about these two methods is available in the Appendix.

The `askAge` method must be very careful about exceptions during input. If the user input is not an integer, `parseInt` will throw a `NumberFormatException`, which `askAge` should catch and deal with appropriately. However, `askAge` cannot recover from the two runtime exceptions possibly thrown by the `next`. If either of those exceptions are thrown, `askAge` should catch them and throw an `AgeException` to indicate that there was some trouble. You may ignore exceptions that could arise from `System.out.println`.

In the class `Age` on the next page, define the method `askAge`, which takes a single argument, the `InputStream` `input`, and returns an `int`. Do not forget to include the appropriate `throws` annotation.

```

import java.io.*;
import java.util.*;

public class Age {

    static class AgeException extends Exception {}

    static final String T1 = "Please input your age:";
    static final String T2 = "Age should be a number between 0 and 120. Please try again.";

    public static int askAge(InputStream input) throws AgeException {
        Scanner scanner = new Scanner(input);
        System.out.println(T1);
        while (true) {
            try {
                int age = Integer.parseInt(scanner.next());
                if (age >= 0 && age <= 120)
                    return age;
            } catch (NumberFormatException exn) {
                // do nothing
            } catch (NoSuchElementException exn) {
                throw (new AgeException());
            } catch (IllegalStateException exn) {
                throw (new AgeException());
            }
            System.out.println(T2);
        }
    }
}

```

Grading guidelines: The emphasis of this problem was the iteration pattern and the treatment of the exceptions. Other sorts of mistakes were generally not deducted.

In grading, we roughly divided the 12 points of this problem among the following tasks:

- 1 correct throws annotation for askAge*
- 3 correct looping structure*
- 2 (each) correctly catching and processing scanner.next() exceptions*
- 2 correctly catching and processing NumberFormatException*
- 1 testing the range of the integer*
- 1 miscellaneous*

3. (12 points) For each of the following scenarios, circle the full type of the data structure that you would use to store the data. This type will be some combination of the generic **Map**, **Set**, and **List** interfaces. You may assume that classes representing each of the objects in the scenarios exist. However, do not assume that these objects store any particular data.

Example: You want to organize the current roster of each NBA **Team** so that **Players** can be found by searching for their team. The answer is that you would circle is:

`Map<Team, Set<Player>>`.

(do not assume, for instance, that a **Team** object contains the set of players on that team.)

- (a) You would like to calculate all of the activities participated in by CSE 120 students, sorted by popularity. You do not need to record what or how many students are involved in each activity. What data structure would you use to store this result?

- i. `List<Activity>`
- ii. `Set<Activity>`
- iii. `Map<Person, Activity>`
- iv. `Map<Person, Set<Activity>>`

Answer: `List<Activity>`

- (b) For each group of CSE 120 students you would like to store all of the activities that they have in common.

- i. `Set<Map<Person, Activity>>`
- ii. `Set<Activity>`
- iii. `Map<Set<Person>, Set<Activity>>`
- iv. `Map<Map<Person,Activity>,Boolean>`

Answer: `Map<Set<Person>, Set<Activity>>`

- (c) In a recipe organizer you would like a data structure that can tell you all recipes that use a particular ingredient.

- i. `Map<Recipe,Ingredient>`
- ii. `Set<Ingredient>`
- iii. `List<Set<Ingredient>,List<Recipe>>`
- iv. `Map<Ingredient,Set<Recipe>>`

Answer: `Map<Ingredient,Set<Recipe>>` (Same as example!)

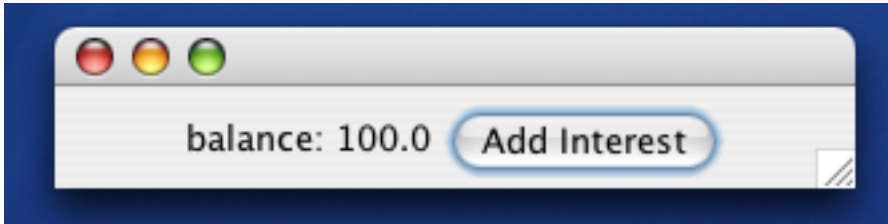
- (d) You would like to create an index of all of the **Files** on a particular website. For each **Word** that appears in any file, you need to calculate what files contain it. You also would like to order the files by how many times they contain an occurrence of that particular word.

- i. `Map<<Set<Word>, List<File>>>`
- ii. `Map<Word, Set<File>>`
- iii. `Map<Word, List<File>>>`
- iv. `Set<Map<File, Word>>`

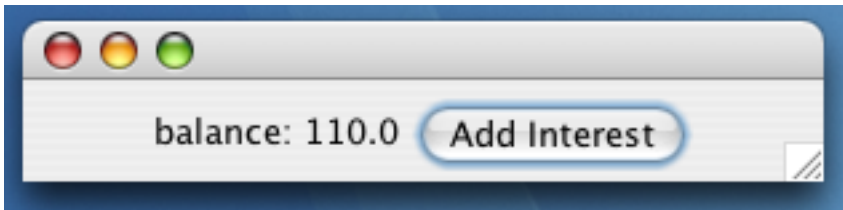
Answer: `Map<Word, List<File>>>`

Grading guidelines: each answer was worth 3 points.

4. (10 points) The last question asks you to complete the implementation of a simple graphical application. This application opens a window that displays an account balance and a button to update that balance with an interest payment.



After pushing the “Add Interest” button, the account balance is increased.



Your task is to complete the implementation of this application by completing the definition of the class `Investment`. This class uses the class `Account`, shown in the Appendix. There are two holes in the `Investment` class and you need to complete both of them:

- (a) This class contains an inner class that implements the `ActionListener` interface—but currently the `actionPerformed` method does nothing.
- (b) The constructor for `Investment` class is not complete—it does not set up the “Add Interest” button.

Please complete the code shown on the next two pages, filling in the indicated sections. For reference, some of the constructors and methods in the classes `JLabel` and `JButton` that we have referred to in lecture, labs, or homework are listed in the Appendix.

```

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

public class Investment extends JFrame {
    final Account account = new Account();

    JButton button;
    JLabel label;

    public class ButtonListener implements ActionListener {

        public void actionPerformed(ActionEvent e) {
            ////////////////////////////////////////////////////
            /// Define method. Your code goes here.
            ////////////////////////////////////////////////////

            account.addInterest();
            label.setText("balance: " + account.getBalance());

            ////////////////////////////////////////////////////
        }
    }
}

```

```

// Constructor for Investment class
Investment() {
    super();

    getContentPane().setLayout(new FlowLayout());

    label = new JLabel();
    label.setText("balance: " + account.getBalance());
    getContentPane().add(label);

    //////////////////////////////////////
    // Set up the button. Your code goes here.
    //////////////////////////////////////

    button = new JButton("Add Interest");
    getContentPane().add(button);
    button.addActionListener(new ButtonListener());

    //////////////////////////////////////

    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(300,60);
    setVisible(true);
}
}

```

Grading guidelines: 4 points for (a) and 6 points for (b). Roughly 2 points for having each of the required lines, with points deducted for lines that are incorrect or are extraneous.

Iterator.java

```
1  interface Iterator<Element> {
2      // Returns true if the iteration has more elements
3      public boolean hasNext();
4
5      // Returns the next element in the iteration
6      // Throws NoSuchElementException if the iteration has no more elements.
7      public Element next();
8
9      // Removes from the underlying collection the last element
10     // returned by the iterator (optional operation).
11     // Throws UnsupportedOperationException if the remove operation is
12     // not supported by this iterator.
13     public void remove();
14 }
```

From java.util.Scanner:

```
public String next()
```

Finds and returns the next complete token from this scanner. A complete token is preceded and followed by input that matches the delimiter pattern. This method may block while waiting for input to scan, even if a previous invocation of hasNext() returned true.

Returns:

the next token

Throws:

NoSuchElementException - if no more tokens are available

IllegalStateException - if this scanner is closed

From java.lang.Integer:

```
public static int parseInt(String s)  
throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002D') to indicate a negative value.

Parameters:

s - a String containing the int representation to be parsed

Returns:

the integer value represented by the argument in decimal.

Throws:

NumberFormatException - if the string does not contain a parsable integer.

Account.java

```
1  public class Account {
2      private double balance;
3      private static final double INITIAL_BALANCE = 100;
4      private static final double INTEREST_RATE = 10;
5
6      public Account () {
7          balance = INITIAL_BALANCE;
8      }
9
10     public double getBalance() { return balance; }
11
12     public void addInterest() {
13         double interest = (balance * INTEREST_RATE) / 100;
14         balance += interest;
15     }
16 }
```

Methods for JLabel:

```
// Creates a JLabel instance containing no text.
JLabel();

// Creates a JLabel instance with the specified text.
JLabel(String s);

// Returns the text string that the label displays
String getText();

// Defines the single line of text that this component will display.
void setText(String text);
```

Methods for JButton:

```
// Creates a button with text
JButton(String s);

// Returns the button's text
String getText();

// Set the button's text
void setText(String s);

// Enables (or disables) the button
void setEnabled(boolean b);

// Adds an action listener to the button
void addActionListener(ActionListener l);
```