CIS 120 — Programming Languages and Techniques

Midterm II

November 12, 2010

Review problems
1. Pages 12 to 14 define a simplified version of the Java Collection interface (the SimpleCollection and SimpleIterator interfaces), together with a concrete implementation using linked lists (the class LinkedSimpleCollection and the auxiliary class Node).

Suppose we wanted to extend LinkedSimpleCollection with a method stutter that duplicates each element in the collection. For example, if the collection is currently holding the three elements "foo", "bar", and "baz" (in that order), then after calling stutter() the collection will hold the six elements "foo", "foo", "bar", "bar", "baz", and "baz" (in that order).

Complete the definition:

```
    public void stutter () {
```
2. Suppose we execute the following program:

```java
public class Main {

    public static <E> void m(SimpleCollection<E> c, E e) {
        c.add(e);
        // HERE
    }

    public static void main(String[] args) {
        SimpleCollection<String> coll1 =
            new LinkedSimpleCollection<String>();
        SimpleCollection<SimpleCollection<String>> coll2 =
            new LinkedSimpleCollection<SimpleCollection<String>>();
        coll1.add("foo");
        coll1.add("bar");
        m(coll2, coll1);
    }
}
```

In the space below, draw a diagram of the memory configuration (both stack and heap) when
execution reaches the line marked **HERE.** (You can omit the parameter **args** to the **main** method.)
3. The listing on page 15 gives an OCaml datatype declaration and some related functions. The listing on pages 16 and 17 gives a set of Java classes providing roughly the same functionality. Ignoring minor differences of syntax between the two languages and focusing on the overall organization of the code, discuss the tradeoffs between the two styles of programming with shapes. Under what circumstances would the first be easier to work with? Under what circumstances would the second be better?
4. What does the following program print?

```java
class C {
    public void foo() {
        try {
            bar();
            System.out.println("here in foo");
        } catch (RuntimeException e) {
            System.out.println("caught in foo");
        }
    }

    public void bar() {
        try {
            baz();
        } catch (RuntimeException e) {
            System.out.println("caught in bar");
            throw e;
        } finally {
            System.out.println("here in bar");
        }
        System.out.println("there in bar");
    }

    public void baz() {
        try {
            if (true) throw new RuntimeException();
        } finally {
            System.out.println("here in baz");
        }
        System.out.println("there in baz");
    }
}

class Exn {
    public static void main(String[] args) {
        (new C()).foo();
    }
}
```

5. Briefly explain the differences between the interfaces InputStream, Reader, and BufferedReader
in the standard Java library.

6. What is an abstract class, and what is it used for?
7. At Penn we play a special version of Quidditch.¹ The game goes on for a pre-determined length of time. Whoever holds the Golden Snitch at the end of the game gets 2012 points. Help determine who is holding the golden snitch and the quaffle at the end of the game. The following Java classes implements the Penn version of the rules, and can be used to simulate a game.

```java
class Ball {
    String description;
    public Ball(String name) {
        description = name;
    }
    public String toString(){
        return description;
    }
}

class Player {
    public Ball ball;
    public Player() { ball = null; }
    public boolean steal(Player p){
        if (p.ball != null && this.ball==null) {
            this.ball = p.ball; p.ball = null; return true;
        } return false;
    }
    public boolean pass(Player p){
        if ( ball != null
            && p.ball==null
            && !ball.toString().equals("Golden Snitch")) {
            p.ball = this.ball; this.ball = null; return true;
        } return false;
    }
}
```

The code on the next page simulates a game.

```
//declare players
Player Harry = new Player(); Player Oliver = new Player();
Player George = new Player(); Player Fred = new Player();
Player Malfoy = new Player(); Player Crabbe = new Player();
Player Goyle = new Player(); Player Miles = new Player();
//declare balls
Ball gs = new Ball("Golden Snitch");
Ball q = new Ball("Quaffle");
//assign starting positions for the balls
```

¹Quidditch is a wizarding sport played on broomsticks. The Penn version uses two balls: a Quaffle and a Golden Snitch. The game starts with two players each holding one of the balls. There are two moves in the game, stealing the ball and passing the ball. See the Java code for more details about how the game is played.
Harry.ball = gs; Malfoy.ball = q;

//begin game
Harry.steal(Malfoy);
Harry.pass(George);
Malfoy.steal(George);
George.pass(Miles);
Malfoy.pass(Goyle);
Crabbe.steal(Miles);
Harry.steal(Goyle);
Harry.pass(Fred);
Malfoy.steal(Harry);
Fred.pass(Harry);

//Game ends.

1. Draw the memory configuration (just the stack and heap, not the workspace or class table) after executing the above declarations and commands.
2. At the end of the game, who has the Golden Snitch?

3. At the end of the game, who has the Quaffle?

8. Complete the following definition of the class IterDouble. Instances of IterDouble are *iterator transformers*—iterators built from other iterators. The IterDouble constructor takes a String iterator wrapped (an instance of the SimpleIterator class defined on page 14) as an argument, and the hasNext and next methods use the corresponding methods of wrapped to do their job. The sequence of strings returned when next is called on the outer IterDouble iterator should be the same as the sequence returned by the inner iterator except that every element of the sequence should be repeated. For example if the inner Iterator produces the sequence

   "a" "b" "c"

then the outer one would produce:

   "a" "a" "b" "b" "c" "c"

```java
class IterDouble implements SimpleIterator<String> {
    // Fields (fill in as needed)

    // Constructor (fill in body)
    IterDouble (SimpleIterator<String> wrapped) {

    }

    // SimpleIterator methods (fill in bodies)
    public boolean hasNext () {

    }
```
public String next () {

// Auxiliary methods (if needed)

}
9. There are also some great review problems on exams from previous offerings of 120. The exact set of topics covered has shifted somewhat this year, so if you look at other parts of previous exams you may find questions involving concepts we haven’t talked about. But you should have all the required background for the following problems:

1. Spring 2010:

   http://www.seas.upenn.edu/~cis120/exams/m1-10a-answers.pdf

   Problems 2, 3, and 5.

2. Fall 2009:

   http://www.seas.upenn.edu/~cis120/exams/practice1/m1-09C-answers.pdf

   Problems 4 and 6. (Problem 6 is especially recommended.)

3. Spring 2009:

   http://www.seas.upenn.edu/~cis120/exams/practice1/m1-09A-answers.pdf

   Problems 1, 3, and 5

4. Fall 2008:

   http://www.seas.upenn.edu/~cis120/exams/practice1/m1-08C-answers.pdf

   Problem 4
For Reference:
SimpleCollection, SimpleIterator, and Node

```java
interface SimpleCollection<E> {
    boolean removeAll(E element);
    boolean add(E element);
    boolean contains(Object o);
    SimpleIterator<E> iterator();
}

interface SimpleIterator<E> {
    boolean hasNext();
    E next();
}

class Node<E> {
    public E element;
    public Node<E> next;
    
    public Node (E element, Node<E> next) {
        this.element = element;
        this.next = next;
    }
}
```
For Reference: LinkedSimpleCollection class

class LinkedSimpleCollection<E> implements SimpleCollection<E> {
    private Node<E> first = null;

    public boolean removeAll(E element) {
        boolean changed = false;
        while (first != null && first.element.equals(element)) {
            first = first.next;
            changed = true;
        }
        if (first == null) return changed;
        Node<E> prev = first;
        for (Node<E> current = first.next; 
             current != null; 
             current = current.next) {
            if (current.element.equals(element)) {
                prev.next = current.next;
                changed = true;
            }
            prev = prev.next;
        }
        return changed;
    }

    public boolean add(E element) {
        Node<E> newnode = new Node<E>(element, first);
        first = newnode;
        return true;
    }

    public boolean contains(Object o) {
        for (Node<E> current = first; current != null; current = current.next) {
            if (current.element == null && o == null)
                || (current.element != null && current.element.equals(o))) {
                return true;
            }
        }
        return false;
    }

    public SimpleIterator<E> iterator() {
        return new LinkedSimpleIterator<E>(first);
    }
}
class LinkedSimpleIterator<E> implements SimpleIterator<E> {
    Node<E> current;
    Node<E> previous;

    public LinkedSimpleIterator (Node<E> first) {
        current = first;
        previous = null;
    }

    public boolean hasNext() {
        return (current != null);
    }

    public E next() {
        if (current == null) {
            throw new java.util.NoSuchElementException();
        } else {
            previous = current;
            E n = current.element;
            current = current.next;
            return n;
        }
    }
}
For Reference: Shapes in OCaml

type double = float

type shape =
    | Point of double ref (* x *)
        * double ref (* y *)
    | Circle of double ref (* x *)
        * double ref (* y *)
        * double ref (* radius *)

let new_point (initX:double) (initY:double) : shape =
    let x = ref initX in
    let y = ref initY in
    Point (x,y)

let new_circle (initX:double) (initY:double) (initR:double) : shape =
    let x = ref initX in
    let y = ref initY in
    let r = ref initY in
    Circle (x,y,r)

let move (s:shape) (dx:double) (dy:double) : unit =
    begin match s with
        | Point (x,y) -> x := !x +. dx; y := !y +. dy
        | Circle (x,y,r) -> x := !x +. dx; y := !y +. dy
    end

let getArea (s:shape) : double =
    begin match s with
        | Point (x,y) -> 0.0
        | Circle (x,y,r) -> 3.14159 *. !r *. !r
    end

let test () =
    let s1 : shape = new_point 5.0 5.0 in
    let s2 : shape = new_circle 0.0 0.0 100.0 in
    moveItALot s1;
    moveItALot s2
interface Shape {
    public void move (double dx, double dy);
    public double getArea ();
}

class Point implements Shape {
    private double x;
    private double y;
    public Point (double initX, double initY) {
        x = initX; y = initY;
    }
    public void move (double dx, double dy) {
        x = x + dx; y = y + dy;
    }
    public double getArea () {
        return 0.0;
    }
}

class Circle implements Shape {
    private double x;
    private double y;
    private double r;
    public Circle (double initX, double initY, double initR) {
        x = initX; y = initY; r = initR;
    }
    public void move (double dx, double dy) {
        x = x + dx; y = y + dy;
    }
    public double getArea () {
        return 3.14159 * r * r;
    }
}

For Reference: Shapes in Java
For Reference: Shapes in Java (continued)

class DoStuff {
    public void moveItALot (Shape s) {
        s.move(3.0,3.0);
        s.move(100.0,1000.0);
        s.move(1000.0,234651.0);
    }

    public void dostuff () {
        Shape s1 = new Point(5.0,5.0);
        Shape s2 = new Circle(0.0,0.0,100.0);
        moveItALot(s1);
        moveItALot(s2);
    }
}