Université d'Ottawa
Faculté de génie

École d'ingénierie et de
technologie de l'information

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

Introduction to Computer Science II (ITI 1221)
Midterm Examination

Instructor: Marcel Turcotte

February 2006, duration: 2 hours

Identification

Student name: _____

Student number: _____ Signature: _____

Instructions

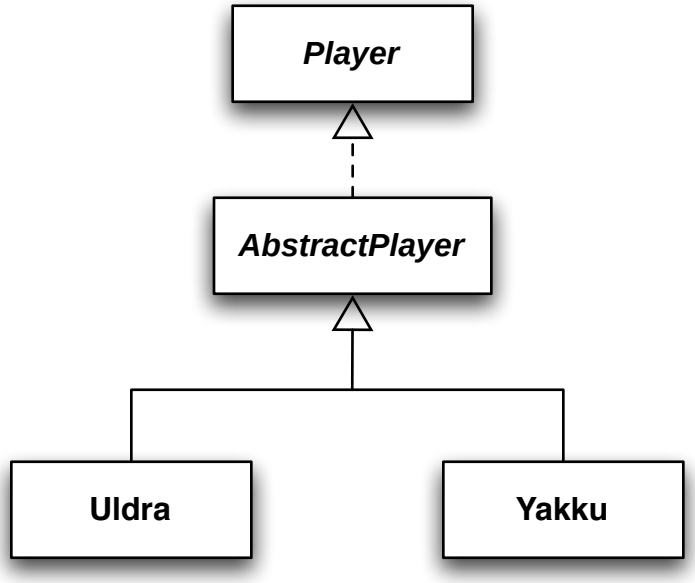
1. This is a closed book examination;
2. No calculators or other aids are permitted;
3. Write comments and assumptions to get partial marks;
4. Beware, poor hand writing can affect grades;
5. Do not remove the staple holding the examination pages together;
6. Write your answers in the space provided. Use the backs of pages if necessary. You may **not** hand in additional pages;

Marking scheme

Question	Maximum	Result
1	30	
2	15	
3	15	
4	10	
5	20	
6	10	
Total	**100**	

Question 1: Inheritance (30 marks)

The company **Electronic Abstractions** (EA) has hired you to work on the development of their latest game. The action is set in the ancient city of Zenda, in the heart of the kingdom of Ruthenia. You will be working on modelling the players (creatures).



Given the above hierarchy and the following information:

- All players of the game have a method `double attack()` that returns a value representing the strength of the attack. `Player` is an interface declaring the method `double attack()`;
- An `AbstractPlayer` defines the characteristics that are common to all the creatures. This abstract class implements the interface `Player`. All the creatures have a `name` (a character `String`) as well as a `life meter` (a `double` ranging from 0.0 to 100.0). The initial value of the `life meter` is 50.0;
- An `Uldra` is a creature (player) that carries a bottle filled with `poison` (a double `value` in the range $0.0 \ldots 100.0$). `Uldra` creatures have a tendency to misplace their bottles (a `boolean` value indicates if this `Uldra` has or not its bottle). The strength of an `attack` is zero if the `Uldra` has no bottle. Otherwise, the strength of an `attack` is 10.0, or the remaining amount of `poison` (which ever value is smaller). The strength of the `attack` reduces the quantity of `poison` left by the same amount;
- A `Yakku` is a creature (player) that inhabits dark and unfriendly places, such as shearing cracks. `Yakku` creatures have `magical powers` (double, range $0.0 \ldots 100.0$, the initial value is 10.0). The strength of the `attack` depends on both its `life meter` and its `magical powers`. Specifically, a weight w is calculated as being the value of the `life meter` divided by 10.0. The weight w is then multiplied by the level of `magical powers` to obtain the strength of the `attack`. An attack reduces both the `life meter` and the `magical powers` by the strength of the `attack`.

Inheritance (continued)

A. In Java, implement the three classes, as well as the interface `Player`. Make sure to include the constructors. For each attribute, write the access methods. The values of the attributes must be valid. If the value of the parameter is less than 0.0 then set the value of the attribute to 0.0. Similarly, if the value of the parameter is larger than the upper limit then set the attribute to its upper limit. Finally, implement the methods `attack`. (20 marks)

Inheritance (continued)

Inheritance (continued)

B. Write a polymorphic class method, `duel`, that has two parameters, both of type `AbstractPlayer`. Let's denote by `first` and `second` the two parameters. While both players are alive (i.e. their `life meter` is greater than zero), the `first` player launches an `attack` onto the `second` one, if the `second` player is still alive, it launches an attack onto the `first` player. When a player `a` launches an `attack` onto a player `b`, you must reduce the value of the `life meter` of `b` by `a.attack()`. Finally, the method displays the name of the player that is still alive. (10 marks)

Question 2: Using a stack (10 marks)

The class `Calculator` (next page) implements a postfix evaluator similar to the one seen in class, as well as in assignment 2. Show the information that will be printed onto the screen when the following two statements are executed:

```
Calculator c = new Calculator();
c.execute( "40  5  2  4  1  ~  print  +  print  ^  print  *  print  -  print" );
```

Notes:

- For this question, there is an implementation of the interface `Stack` called `LinkedStack`. See Appendix A;
- The implementation of the classes `Token` and `Reader` can be found in Appendix B and C.

Using a stack (continued)

```
public class Calculator {
    private Stack operands = new LinkedStack();
    public void execute( String program ) {
        Reader r = new Reader( program );
        while ( r.hasMoreTokens() ) {
            Token t = r.nextToken();
            if ( ! t.isSymbol() ) {
                operands.push( t );
            } else if ( t.sValue().equals( "+" ) ) {
                Token b = (Token) operands.pop();
                Token a = (Token) operands.pop();
                Token res = new Token( a.iValue() + b.iValue() );
                operands.push( res );
            } else if ( t.sValue().equals( "-" ) ) {
                Token b = (Token) operands.pop();
                Token a = (Token) operands.pop();
                Token res = new Token( a.iValue() - b.iValue() );
                operands.push( res );
            } else if ( t.sValue().equals( "*" ) ) {
                Token b = (Token) operands.pop();
                Token a = (Token) operands.pop();
                Token res = new Token( a.iValue() * b.iValue() );
                operands.push( res );
            } else if ( t.sValue().equals( "~" ) ) {
                Token o = (Token) operands.pop();
                Token res = new Token( - o.iValue() );
                operands.push( res );
            } else if ( t.sValue().equals( "^" ) ) {
                Token b = (Token) operands.pop();
                Token a = (Token) operands.pop();
                Token res = new Token( (int) Math.pow( a.iValue(), b.iValue() ) );
                operands.push( res );
            } else if ( t.sValue().equals( "print" ) ) {
                System.out.println( "-top-" );
                Stack tmp = new LinkedStack();
                while ( ! operands.isEmpty() ) {
                    t = (Token) operands.pop();
                    System.out.println( t );
                    tmp.push( t );
                }
                while (! tmp.isEmpty()) {
                    operands.push( tmp.pop() );
                }
                System.out.println( "-bottom-" ); System.out.println();
            }
        }
    }
}
```

Question 3: DynamicArrayStack (15 marks)

The class `DynamicArrayStack` below uses a technique seen in class, as well as in assignment 2, to expand, or shrink, its physical size whenever the data structure is full, or uses too much memory.

- `DynamicArrayStack` uses an array of references of type `Object` to store the elements of this stack;
- The initial capacity of this array is given by the first parameter of the constructor;
- The physical size of the array is increased by a fixed amount (`increment`) when the method `void push( Object elem )` is called and the array is full;
- The physical size of the array is decreased by a fixed amount (`increment`) during a call to the method `Object pop()` if the number of free cells becomes `increment` or more;
- The `increment` is given by the second parameter of the constructor;
- The variable `size` indicates the number of elements stored in the array as well as the first free cell of the array.

A. Correct at least 5 mistakes (logic, runtime or compile-time errors) in the partial implementation. (5 marks)

B. Complete the partial implementation of the class `DynamicArrayStack` given the above information. (10 marks)

```
public class DynamicArrayStack extends Stack {

    // Instance variables
    private Object[] elems;      // Stores the elements of this stack
    private int size = 0;        // Also designates the first free cell
    private final int capacity;  // Memorizes the initial capacity
    private final int increment; // Used to increase/decrease the size

    public DynamicArrayStack( int capacity, int increment ) {
        Object[] elems = new Object[ capacity ];
        capacity = capacity;
    }

    // Returns true if this stack is empty;
    public boolean isEmpty() {
        return size == 0;
    }

    // Continues on the next page ...
```

DynamicArrayStack (continued)

```
// Puts an element onto the top of this stack

public void push( Object element ) {

    if ( ____________________________________________________ ) {
        increaseSize();
    }

    elems[ size ] = element;
    size = size + 1;
}

// Increases the size of the array

private void increaseSize() {
    Object[] newElems;

    newElems = new Object[ ________________________________________ ];

    // Copying all the elements to the new array
    newElems = elems;

    // Replacing elems with the new and larger array

    ________________________________________;
}

// Continues on the next page ...
```

DynamicArrayStack (continued)

```
    // Removes and returns the top element of this stack

    public Object pop() {

        if ( ____________________________________________________ ) {
            decreaseSize();
        }

        Object saved = elems[ size ];
        size = size - 1;

        // Scrubbing the memory!
        elems[ size ] = __________;


    }

    // Decreases the size of the array

    private void decreaseSize() {

        int newSize = elems.length - increment;

        if ( newSize < capacity ) {
            newSize = capacity;
        }

        Object[] newElems;
        newElems = new Object[ newSize ];

        for ( int i=0; i<size; i++ ) {
            newElems[ i ] = elems[ i ];
        }

        // Replacing elems with the new/smaller array

        ________________________________________;
    }
} // end of DynamicArrayStack
```

Question 4: Overriding equals (10 marks)

Every class inherits the method `public boolean equals( Object other )` from the class `Object`. In the class `CombinationLock` below, override this definition of the method `equals`. Two locks are `equals` if their status are "equals" and both `CombinationLock` have the same combination (defined by the variables `first`, `second` and `third`). Otherwise, the method returns `false`. No values are considered illegal; in particular, the method should be handling `null` values. The definition of the class `State` can be found in Appendix D.

```
public class CombinationLock {

    // Instance variables
    private int first;
    private int second;
    private int third;
    private State status;

    // Constructor
    public CombinationLock( int first, int second, int third ) {
        this.first = first;
        this.second = second;
        this.third = third;
        status = new State();
    }

    public boolean equals(                        ) {









    } // end of equals
} // end of CombinationLock
```

Question 5: LinkedPair (20 marks)

A `Pair` represents a group of two objects. Here, the objects of a `Pair` are `Comparable` objects. The interface `Pair` defines two methods: `public Comparable getFirst()` returns a reference to the first object of the `Pair` while `public Comparable getSecond()` returns a reference to the second object of the `Pair`. The implementation of the interfaces `Pair` and `Comparable` can be found in Appendix E and F.

`LinkedPair` (next page) is an implementation of the interface `Pair` that uses objects of the class `Elem` to store the objects of the `Pair`. Specifically, in the class `LinkedPair`, the instance variable `first` designates an object of the class `Elem` used to store the first object of the `Pair`. The instance variable `next` of the object designated by `first` points at an object of the class `Elem` used to store the second object of the `Pair`. Adding new instance variables, either to the class `LinkedPair` or `Elem`, is strictly prohibited.

A. Draw the memory diagram representing the content of the memory after the execution of the following statement. The implementation of the class `Name` can be found in Appendix G. (8 marks)

```
Pair p = new LinkedPair(new Name("Joseph", "Rotblat"), new Name("Wangari", "Maathai"));
```

B. For the partial implementation of the class `LinkedPair`, write the instance method `public sort()` that sorts the (two) elements of a `LinkedPair`. The implementation of the method `sort` **must** change the order of the elements (objects of the class `Elem`) — **exchanging the values of the nodes is not acceptable**. (10 marks)

C. Complete the implementation of the method `String toString()`. It returns a `String` representation of the pair starting with "(", followed by the `String` representation of the first object of the pair, followed by ",", followed by the `String` representation of the second object of the pair, followed by ")". (2 marks)

```
public class LinkedPair implements Pair {

    private static class Elem {
        private Comparable value;
        private Elem next;
        private Elem( Comparable value, Elem next ) {
            this.value = value;
            this.next = next;
        }
    }

    private Elem first;

    public LinkedPair( Comparable a, Comparable b ) {
        if ( a == null || b == null )
             throw new IllegalArgumentException( "null(s) value(s)" );

        first = new Elem( a, new Elem( b, null ) );
    }

    public void sort() {









    } // end of sort
    public String toString() {


    }
} // end of LinkedStack
```

Question 6: Short answer questions (10 marks)

A. Define a new checked exception called `ZendaIOException`. This should be a specialised `IOException`. (4 marks)

B. Consider the class `Zenda` on the next page. Modify the declaration of all three methods so that **only** the exceptions needing to be declared are actually declared by the methods (i.e. remove those exceptions that can be removed without causing a compile-time error). Note that `FileNotFoundException` and `IllegalArgumentException` are checked and unchecked exception types respectively. (4 marks)

C. Explain why a runtime exception will occur if the following statements are executed. (2 marks)

```
public class Manager {
    private Object[] elems;
    public Manager( int capacity ) {
        if ( elems.length == 0 ) {
            elems = new Object[ capacity ];
        }
    }
    public static void main( String[] args ) {
        Manager m = new Manager( 100 );
    }
}
```

Short answer questions (continued)

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class Zenda {

    private static void getConfig( String name )
        throws ZendaIOException, FileNotFoundException, IllegalArgumentException {

        if ( name == null ) {
            throw new IllegalArgumentException( "null value" );
        }

        FileInputStream input;

        try {

            input = new FileInputStream( name );

        } catch ( FileNotFoundException e ) {
            throw new ZendaIOException( "file not found: " + name );
        }
    }

    private static void init()
        throws ZendaIOException, FileNotFoundException, IllegalArgumentException {

        getConfig( "config.dat" );
    }

    public static void main( String[] args )
        throws ZendaIOException, FileNotFoundException, IllegalArgumentException {

        try {

            init();

        } catch ( ZendaIOException e ) {
            System.err.println( "ERROR READING CONFIG FILE" );
            e.printStackTrace();
            System.exit( 1 );
        }
    }
}
```

A Stack

```
public interface Stack {

    /**
     * Tests if this Stack is empty.
     *
     * @return true if this Stack is empty; and false otherwise.
     */

    public abstract boolean isEmpty();

    /**
     * Returns a reference to the top element; does not change
     * the state of this Stack.
     *
     * @return The top element of this stack without removing it.
     */

    public abstract Object pop();

    /**
     * Puts an element onto the top of this stack.
     *
     * @param element the element be put onto the top of this stack.
     */

    public abstract void push( Object element );

}
```

B Token

```
public class Token {
    private static final int INTEGER = 1;
    private static final int SYMBOL = 2;
    private int iValue;
    private String sValue;
    private int type;
    public Token( int iValue ) {
        this.iValue = iValue;
        type = INTEGER;
    }
    public Token( String sValue ) {
        this.sValue = sValue;
        type = SYMBOL;
    }
    public int iValue() {
        // pre-condition: this Token represents an integer value
        return iValue;
    }
    public String sValue() {
        // pre-condition: this Token represents a symbol
        return sValue;
    }
    public boolean isInteger() {
        return type == INTEGER;
    }
    public boolean isSymbol() {
        return type == SYMBOL;
    }
    public String toString() {
        switch ( type ) {
        case INTEGER:
            return "INTEGER: " + iValue;
        case SYMBOL:
            return "SYMBOL: " + sValue;
        default:
            return "INVALID";
        }
    }
}
```

C Reader

```
import java.util.StringTokenizer;

public class Reader {

    private StringTokenizer st;

    public Reader( String s ) {
        st = new StringTokenizer( s );
    }
    public boolean hasMoreTokens() {
        return st.hasMoreTokens();
    }

    public Token nextToken() {

        String t = st.nextToken();

        if ( "true".equals( t ) )
            return new Token( true );

        if ( "false".equals( t ) )
            return new Token( false );

        try {
            return new Token( Integer.parseInt( t ) );

        } catch ( NumberFormatException e ) {

            return new Token( t );

        }
    }
}
```

D State

```
public class State {

    // Constants
    private static final int IS_OPEN = 0;
    private static final int IS_LOCKED = 1;
    private static final int IS_DEACTIVATED = 2;

    // Instance variable
    private int status = IS_OPEN;

    // Access methods
    public boolean isOpen() {
        return status == IS_OPEN;
    }

    public boolean isLocked() {
        return status == IS_LOCKED;
    }

    public boolean isDeactivated() {
        return status == IS_DEACTIVATED;
    }

    public void open() {
        status = IS_OPEN;
    }

    public void lock() {
        status = IS_LOCKED;
    }

    public void deactivate() {
        status = IS_DEACTIVATED;
    }

    // Overwrites the method equals.
    // You can use the method equals, however, its implementation has been withheld.
}
```

E Pair

```
public interface Pair {

    /**
      * Returns the first element of this Pair.
      *
      * @return the first element of this Pair.
      */

    public abstract Comparable getFirst();

    /**
      * Returns the second element of this Pair.
      *
      * @return the second element of this Pair.
      */

    public abstract Comparable getSecond();

}
```

F Comparable

```
public interface Comparable {

    /**
     * Compares this object with the specified object for order.  Returns a
     * negative integer, zero, or a positive integer as this object is less
     * than, equal to, or greater than the specified object.<p>
     *
     * @param   o the Object to be compared.
     * @return  a negative integer, zero, or a positive integer as this object
     *          is less than, equal to, or greater than the specified object.
     *
     * @throws ClassCastException if the specified object's type prevents it
     *         from being compared to this Object.
     */

    public abstract int compareTo( Object o );

}
```

G Name

```
public class Name implements Comparable {

    // Instance variables
    private String firstName;
    private String lastName;

    // Constructor
    public Name( String firstName, String lastName ) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    // Access methods
    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    // Implements the method compareTo( Object obj )
    public int compareTo( Object obj ) {
        Name other = (Name) obj;
        int result = lastName.compareTo( other.lastName );
        if ( result == 0 ) {
            result = firstName.compareTo( other.fistName );
        }
        return result;
    }

    // Overwrites the method toString
    public String toString() {
        return firstName + " " + lastName;
    }
}
```

(blank space)