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 informatique II (ITI 1621)
Examen Intra

Instructeur: Marcel Turcotte

Février 2006, durée: 2 heures

Identification

Nom, prénom : _____

Numéro d'étudiant : _____ Signature : _____

Consignes

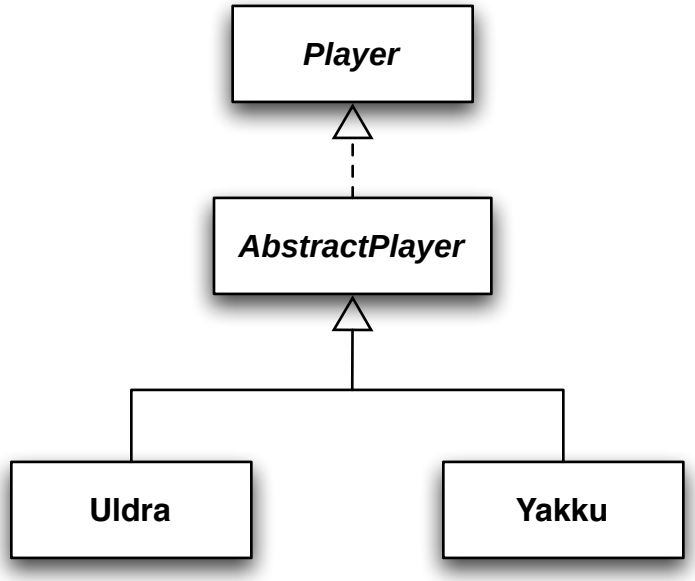
1. Livres fermés ;
2. Sans calculatrice ou toute autre forme d'aide ;
3. Répondez sur ce questionnaire, utilisez le verso des pages si nécessaire, mais vous ne pouvez remettre aucune page additionnelle ;
4. Écrivez lisiblement, votre note en dépend ;
5. Commentez vos réponses ;
6. Ne retirez pas l'agrafe.

Barème

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

Question 1 : Héritage (30 points)

Vous avez été embauché par la compagnie **Electronic Abstractions** (EA) afin de travailler au développement de leur tout nouveau jeu vidéo. L'action se déroule dans l'ancienne cité de Zenda, au coeur du royaume de la Ruthénie. Vous travaillerez au développement d'une hiérarchie de classes afin de modéliser les joueurs (créatures).



Étant donné la hiérarchie de classes ci-haut et les informations suivantes :

- Tous les joueurs de ce jeu ont une méthode `double attack()` retournant une valeur représentant la force de cette attaque. L'interface `Player` déclare la méthode `double attack()` ;
- La classe `AbstractPlayer` définit les caractéristiques communes à l'ensemble des créatures. Cette classe abstraite réalise l'interface `Player`. Toutes les créatures ont un `nom` (une chaîne de caractères) ainsi qu'un `indicateur de niveau de vie` (`double` dont l'intervalle de valeurs est 0.0 à 100.0). La valeur initiale de l'`indicateur de niveau de vie` est 50.0.
- Un `Uldra` est une créature (joueur) qui déambule avec une bouteille remplie de `poison` (double, valeur de 0.0 à 100.0). L'`Uldra` est une créature maladroite qui égare souvent sa bouteille (un booléen indique si cette créature possède ou non une bouteille). La force de son attaque est zéro (0.0) si cet `Uldra` n'a pas sa bouteille. Sinon, la force de l'attaque est 10.0, ou la quantité de poison restante si la quantité de poison est inférieure à 10.0. L'attaque réduit la quantité de `poison` par la même valeur ;
- Le `Yakku` est une créature (joueur) habitant les endroits sombres, tels que les cavernes et les crevasses. Ces créatures ont des pouvoirs magiques (double, intervalle $0.0 \ldots 100.0$) dont la valeur initiale est 10.0. La force de son attaque dépend à la fois de son indicateur de niveau de vie et de ses pouvoirs magiques. Spécifiquement, un poids w, dont la valeur est celle de l'indicateur de niveau de vie divisée 10.0, est multiplié par la valeur des pouvoirs magiques afin de déterminer la force de l'attaque. L'attaque réduit les niveaux de vie et de magie d'une quantité proportionnelle à la force de l'attaque.

Héritage (suite)

A. En Java, vous devez implémenter les trois classes décrites ci-haut, ainsi que l'interface `Player`. Assurez-vous d'y inclure tous les constructeurs. Pour chaque attribut, vous devez écrire les méthodes d'accès. Assurez-vous que les valeurs des attributs sont valides. Si la valeur du paramètre est plus petite que 0.0, alors, affectez la valeur 0.0. Si la valeur excède la limite supérieure, affectez la valeur de la limite supérieure. Finalement, vous devez implémenter les méthodes `double attack()`. (20 points)

Héritage (suite)

Héritage (suite)

B. Implémentez la méthode de classe polymorphique `duel`, ayant deux paramètres, tous deux de type `AbstractPlayer`. Soient `first` et `second` les identificateurs de ces deux paramètres. Tant que les deux joueurs sont en vie (i.e. leur indicateur de niveau de vie est supérieur à zéro), le premier joueur attaque le second, si le second joueur est toujours en vie, il lance une attaque à son tour contre le premier joueur. Lorsqu'un joueur `a` attaque un joueur `b`, vous devez réduire le niveau de vie du joueur `b` de la quantité `a.attack()`. Finalement, la méthode affiche le nom du joueur qui est toujours vivant. (10 points)

Question 2 : Utilisation d'une pile (10 points)

La classe `Calculator` (page suivante) implémente un interprète pour des expressions en notation postfixe semblable à celui étudié en classe, ainsi qu'au devoir 2. Donnez le contenu de la sortie suite à l'exécution des deux énoncés suivants :

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

Notes :

- Pour cette question, la classe `LinkedStack` réalise l'interface `Stack`. Consultez la section A ;
- Vous trouverez l'implémentation des classes `Token` et `Reader` aux sections B et C.

Utilisation d'une pile (suite)

```
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 points)

La classe `DynamicArrayStack` ci-bas utilise une technique vue en classe, ainsi qu'au devoir 2, afin d'accroître, ou de diminuer, la taille physique de la structure de données lorsque la structure est remplie, ou encore utilise trop d'espace mémoire.

- `DynamicArrayStack` utilise un tableau dont les références sont de type `Object` afin de sauvegarder les éléments de cette pile ;
- La capacité initiale du tableau est spécifiée à l'aide du premier paramètre du constructeur ;
- La taille du tableau doit être agrandie d'une quantité fixe (`increment`) lors d'un appel à la méthode `void push( Object elem )` lorsque le tableau est rempli à capacité ;
- La taille du tableau doit être diminuée d'une quantité fixe (`increment`) lors d'un appel à la méthode `Object pop()` lorsque le nombre de cellules vides devient `increment` ou plus ;
- La valeur de l'`increment` est obtenue à l'aide du second paramètre du constructeur ;
- La variable `size` indique la taille logique de la pile ainsi que la position de la première cellule libre du tableau.

A. Vous devez corriger au moins 5 erreurs (des erreurs de logique, des erreurs d'exécution et des erreurs de compilation) dans cette implémentation partielle ; (5 points)

B. Complétez l'implémentation partielle de la classe `DynamicArrayStack` compte tenu des informations ci-haut. (10 points)

```
public class DynamicArrayStack extends Stack {

    // Variables d'instance
    private Object[] elems;      // Les éléments de cette pile
    private int size = 0;        // Désigne aussi la première cellule vide
    private final int capacity;  // Mémorise la capacité initiale
    private final int increment; // Pour accoître/décroitre la taille

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

    // Vrai si cette pile est vide
    public boolean isEmpty() {
        return size == 0;
    }

    // Voir page suivante ...
```

DynamicArrayStack (suite)

```
// Ajout d'un élément sur le dessus de cette pile

public void push( Object element ) {

    if ( ________________________________________________ ) {
        increaseSize();
    }

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

// Accroître la taille de cette pile

private void increaseSize() {
    Object[] newElems;

    newElems = new Object[ ______________________________________ ];

    // Copier tous les éléments dans un nouveau tableau
    newElems = elems;

    // Remplacer elems par un nouveau tableau plus grand

    ______________________________________;
}

// Voir page suivante ...
```

DynamicArrayStack (suite)

```
    // Retire et retourne l'élément du dessus de la pile

    public Object pop() {

        if ( ________________________________________________ ) {
            decreaseSize();
        }

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

        // Coup de main au gc ("memory scrubbing")
        elems[ size ] = __________;


    }

    // Diminuer la taille du tableau

    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 ];
        }

        // Remplacer elems par tableau plus petit

        ________________________________________;
    }
} // Fin de DynamicArrayStack
```

Question 4 : Redéfinition de la méthode equals (10 points)

Toute classe hérite de la méthode `public boolean equals( Object other )` de la classe `Object`. Dans la classe `CombinationLock` ci-bas, vous devez redéfinir cette méthode. Deux verrous (CombinationLock) sont égaux si les "status" sont égaux et que leurs combinaisons sont les mêmes (la combinaison est définie par les variables `first`, `second` et `third`). Sinon, la méthode doit retourner `false`. Il n'y a aucune valeur illégale ; en particulier, la méthode doit traiter les valeurs `null`. La définition de la classe `State` se trouve à la section D.

```
public class CombinationLock {

    // Variables d'instance
    private int first;
    private int second;
    private int third;
    private State status;

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

    public boolean equals(                          ) {









    } // Fin de equals
} // Fin de CombinationLock
```

Question 5 : LinkedPair (20 points)

Une `Pair` regroupe deux objets. Ici, les objets d'une `Pair` sont `Comparable`. L'interface `Pair` définit deux méthodes : `public Comparable getFirst()` retourne une référence vers le premier objet de cette `Pair` alors que `public Comparable getSecond()` retourne une référence vers le second objet de la `Pair`. L'implémentation des interfaces `Pair` et `Comparable` se trouve aux sections E et F.

`LinkedPair` (page suivante) est une implémentation de l'interface `Pair` utilisant des objets de la classe `Elem` afin de sauvegarder les objets de cette `Pair`. Spécifiquement, la classe `LinkedPair` possède une variable d'instance, `first`, désignant un objet de la classe `Elem` que l'on utilise afin de sauvegarder le premier objet de cette paire. La variable d'instance `next` de l'objet désigné par `first` désigne un objet de la classe `Elem` utilisé afin de sauvegarder le second objet de cette paire. L'ajout de nouvelles variables d'instance, aux classes `LinkedPair` ou `Elem`, est strictement interdit.

A. Dessinez le diagramme de mémoire représentant le contenu de la mémoire suite à l'exécution de cet énoncé. L'implémentation de la classe `Name` se trouve à la section G du test. (8 points)

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

B. Ajoutez à l'implémentation partielle de la classe `LinkedPair` une méthode d'instance, `public sort()`, afin de trier ces (deux) objets. L'implémentation de cette méthode **doit** changer l'ordre des éléments (objets de la classe `Elem`) — **échanger les valeurs des noeuds n'est pas une solution acceptable**. (10 points)

C. Complétez l'implémentation de la méthode `String toString()`. Elle retourne une chaîne débutant par "(", suivi de la chaîne représentant le premier objet de la paire, suivi de ",", suivi de la chaîne représentant le second objet de la paire, suivi de ")". (2 points)

```
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() {









    } // Fin de sort
    public String toString() {


    }
} // Fin de LinkedStack
```

Question 6 : Réponses brèves (10 points)

A. Vous devez définir un nouveau type d'exceptions dites «checked» nommé `ZendaIOException`. `ZendaIOException` est une spécialisation de la classe `IOException`. (4 points)

B. Pour la classe `Zenda` se trouvant à la page suivante. Modifiez la déclaration des trois méthodes afin qu'elles ne déclarent que les exceptions nécessaires (omettez les exceptions qui pourraient être omises sans causer une erreur de compilation). `FileNotFoundException` et `IllegalArgumentException` sont dites «checked» et «unchecked» respectivement. (4 points)

C. Expliquez pourquoi l'exécution des énoncés suivants causera une erreur. (2 points)

```
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 );
    }
}
```

Réponses brèves (suite)

```
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 {

    /**
     * Vérifie si cette pile est vide.
     *
     * @return true si cette pile est vide; et false sinon.
     */

    public abstract boolean isEmpty();

    /**
     * Retourne une référence vers l'élément du dessus; ne change
     * pas l'état de cette pile.
     *
     * @return l'élément du dessus sans le retirer.
     */

    public abstract Object pop();

    /**
     * Ajoute un élément sur la pile.
     *
     * @param l'élément à ajouter sur la pile.
     */

    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: ce jeton représente un entier
        return iValue;
    }
    public String sValue() {
        // pre-condition: ce jeton représente un symbole
        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 {

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

    // Variables d'instance
    private int status = IS_OPEN;

    // Méthodes d'accès
    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;
    }

    // Redéfinition de la méthode equals.
    // Vous pouvez utiliser la méthode equals de cette classe, sa
    // définition vous est cependant cachée.
}
```

E Pair

```
public interface Pair {

    /**
      * Retourne le premier élément de la Pair.
      *
      * @return le premier élément de la Pair.
      */

    public abstract Comparable getFirst();

    /**
      * Retourne le second élément de la Pair.
      *
      * @return le second élément de la Pair.
      */

    public abstract Comparable getSecond();

}
```

F Comparable

```
public interface Comparable {

    /**
     * Compare cet objet et celui spécifié en paramètre. Retourne un entier
     * négatif, zéro, ou un entier positive selon que cet objet est plus petit
     * que, égal à, ou plus grand que l'objet spécifié.<p>
     *
     * @param   o l'objet a comparer.
     * @return  un entier négatif, zéro, ou un entier positif selon que cet
     *          objet est plus petit, égal à, ou plus grand que l'objet
     *          spécifié.
     *
     * @throws ClassCastException si le type de l'objet spécifié ne permet
     *         pas la comparaison avec cet objet.
     */

    public abstract int compareTo( Object o );

}
```

G Name

```
public class Name implements Comparable {

    // Variables d'instance
    private String firstName;
    private String lastName;

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

    // Méthodes d'accès
    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    // Implémente 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;
    }

    // Redéfinition de la méthode toString
    public String toString() {
        return firstName + " " + lastName;
    }
}
```

(page blanche)