Data Structures and Algorithms

COMP 103 2019-20 Semester 2

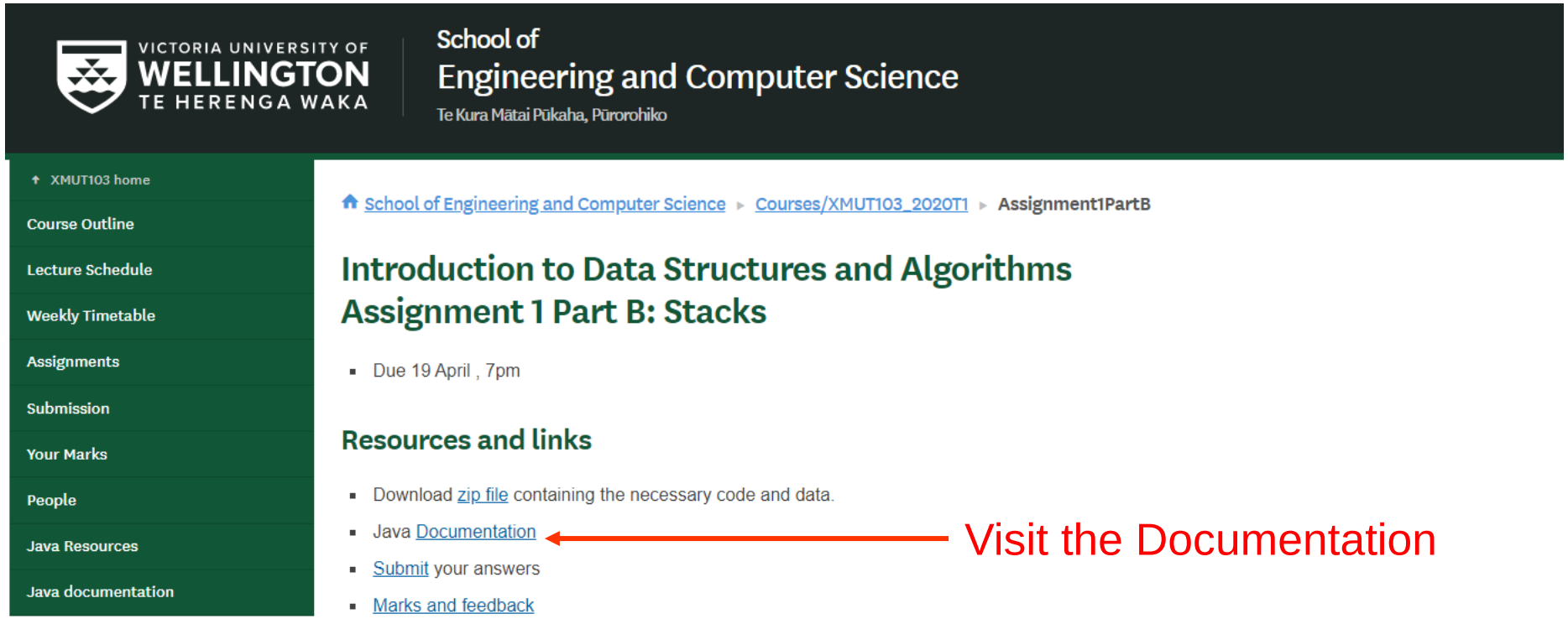
Lecture 04a

Dr. Kerese Manueli

kerese.manueli@ecs.vuw.ac.nz

Victoria University of Wellington

https://ecs.wgtn.ac.nz/Courses/XMUT103_2020T1/Assignment1PartB



VICTORIA UNIVERSITY OF WELLINGTON TE HERENGA WAKA

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

- XMUT103 home
- Course Outline
- Lecture Schedule
- Weekly Timetable
- Assignments
- Submission
- Your Marks
- People
- Java Resources
- Java documentation

School of Engineering and Computer Science ▸ Courses/XMUT103_2020T1 ▸ Assignment1PartB

Introduction to Data Structures and Algorithms
Assignment 1 Part B: Stacks

- Due 19 April , 7pm

Resources and links

- Download zip file containing the necessary code and data.
- Java Documentation ← Visit the Documentation
- Submit your answers
- Marks and feedback
- Part A of the assignment

What To Hand In

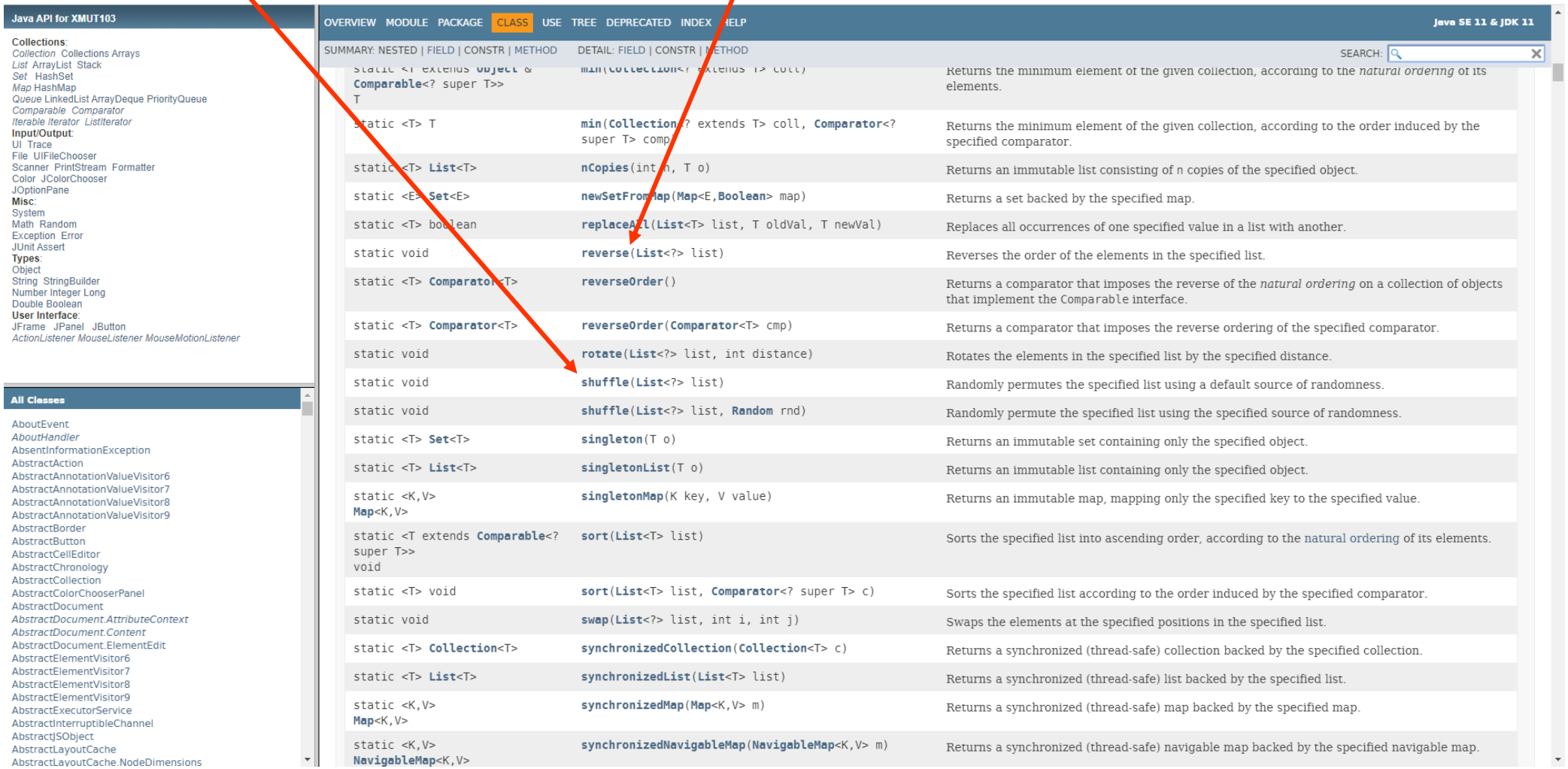
- `SlideShow.java`
- `Sokoban.java` (and any additional files you change or create for `Sokoban`)

Do not rename these files.

Remember to submit all of these files. When you have submitted them, check that you can read the files listed on the submission page, and complete the submission process.

Collections.shuffle(*name of list*)

Collections.reverse(*name of list*)



Java API for XMUT103

Collections:
Collection Collections Arrays
List ArrayList Stack
Set HashSet
Map HashMap
Queue LinkedList ArrayDeque PriorityQueue
Comparable *Comparator*
Iterable *Iterator* *ListIterator*
Input/Output:
UI Trace
File UIFileChooser
Scanner PrintStream Formatter
Color JColorChooser
JOptionPane
Misc:
System
Math Random
Exception Error
JUnit Assert
Types:
Object
String StringBuilder
Number Integer Long
Double Boolean
User Interface:
JFrame JPanel JButton
ActionListener *MouseListener* *MouseMotionListener*

All Classes

AboutEvent
AboutHandler
AbsentInformationException
AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractAnnotationValueVisitor8
AbstractAnnotationValueVisitor9
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractElementVisitor8
AbstractElementVisitor9
AbstractExecutorService
AbstractInterruptibleChannel
AbstractJSObject
AbstractLayoutCache
AbstractLayoutCache.NodeDimensions

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java SE 11 & JDK 11

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH:

static <T extends Object & Comparable<? super T>> T	min(Collection<? extends T> coll)	Returns the minimum element of the given collection, according to the *natural ordering* of its elements.
static <T> T	min(Collection<? extends T> coll, Comparator<? super T> comp)	Returns the minimum element of the given collection, according to the order induced by the specified comparator.
static <T> List<T>	nCopies(int n, T o)	Returns an immutable list consisting of n copies of the specified object.
static <E> Set<E>	newSetFromMap(Map<E,Boolean> map)	Returns a set backed by the specified map.
static <T> boolean	replaceAll(List<T> list, T oldVal, T newVal)	Replaces all occurrences of one specified value in a list with another.
static void	reverse(List<?> list)	Reverses the order of the elements in the specified list.
static <T> Comparator<T>	reverseOrder()	Returns a comparator that imposes the reverse of the *natural ordering* on a collection of objects that implement the Comparable interface.
static <T> Comparator<T>	reverseOrder(Comparator<T> cmp)	Returns a comparator that imposes the reverse ordering of the specified comparator.
static void	rotate(List<?> list, int distance)	Rotates the elements in the specified list by the specified distance.
static void	shuffle(List<?> list)	Randomly permutes the specified list using a default source of randomness.
static void	shuffle(List<?> list, Random rnd)	Randomly permute the specified list using the specified source of randomness.
static <T> Set<T>	singleton(T o)	Returns an immutable set containing only the specified object.
static <T> List<T>	singletonList(T o)	Returns an immutable list containing only the specified object.
static <K,V> Map<K,V>	singletonMap(K key, V value)	Returns an immutable map, mapping only the specified key to the specified value.
static <T extends Comparable<? super T>> void	sort(List<T> list)	Sorts the specified list into ascending order, according to the natural ordering of its elements.
static <T> void	sort(List<T> list, Comparator<? super T> c)	Sorts the specified list according to the order induced by the specified comparator.
static void	swap(List<?> list, int i, int j)	Swaps the elements at the specified positions in the specified list.
static <T> Collection<T>	synchronizedCollection(Collection<T> c)	Returns a synchronized (thread-safe) collection backed by the specified collection.
static <T> List<T>	synchronizedList(List<T> list)	Returns a synchronized (thread-safe) list backed by the specified list.
static <K,V> Map<K,V>	synchronizedMap(Map<K,V> m)	Returns a synchronized (thread-safe) map backed by the specified map.
static <K,V> NavigableMap<K,V>	synchronizedNavigableMap(NavigableMap<K,V> m)	Returns a synchronized (thread-safe) navigable map backed by the specified navigable map.

https://ecs.wgtn.ac.nz/Courses/XMUT103_2020T1/Assignment1PartB

Undo for Sokoban (Weight: 1/3)

Sokoban is a computer game of the puzzle variety that was created in 1980, and is available on many computer platforms. The game involves controlling a worker in a warehouse who has to push boxes around the warehouse to get them onto their destination spots. The worker can only push one box at a time, and cannot pull boxes. It is easy for the player to get stuck in a deadlock where it is no longer possible to move some of the boxes. Some of the levels are extremely difficult to solve.

You can find out more about the Sokoban game from the web.

We have provided a simple implementation of the Sokoban game with four levels. You can control the worker with buttons or keys.

Our implementation of Sokoban is frustrating because if you make a mistake, you have to restart the game from the beginning. The game desperately needs an *undo* facility, so that you can undo actions if you discover you have made a mistake. It should be possible to undo all actions, all the way back to the beginning of a level.

For this assignment, you need to add an **undo** button to `Sokoban`. It should use a `Stack` that contains the history of actions since the beginning of the current level. Every time the player performs a (successful) action, the action should be recorded on the stack. Every time the player clicks the `undo` button, the program should pop the top action from the history and "undo" it: if it was a move, the worker should be moved back (in the opposite direction of the recorded direction); if it was a push, the worker should be moved back, along with the box that was pushed.

The amount of code you have to write is quite small, but to work out what is needed and where to put it, you will have to read and understand a lot (if not all) of the Sokoban program. This will not be trivial and you should expect to spend time reading the program carefully.

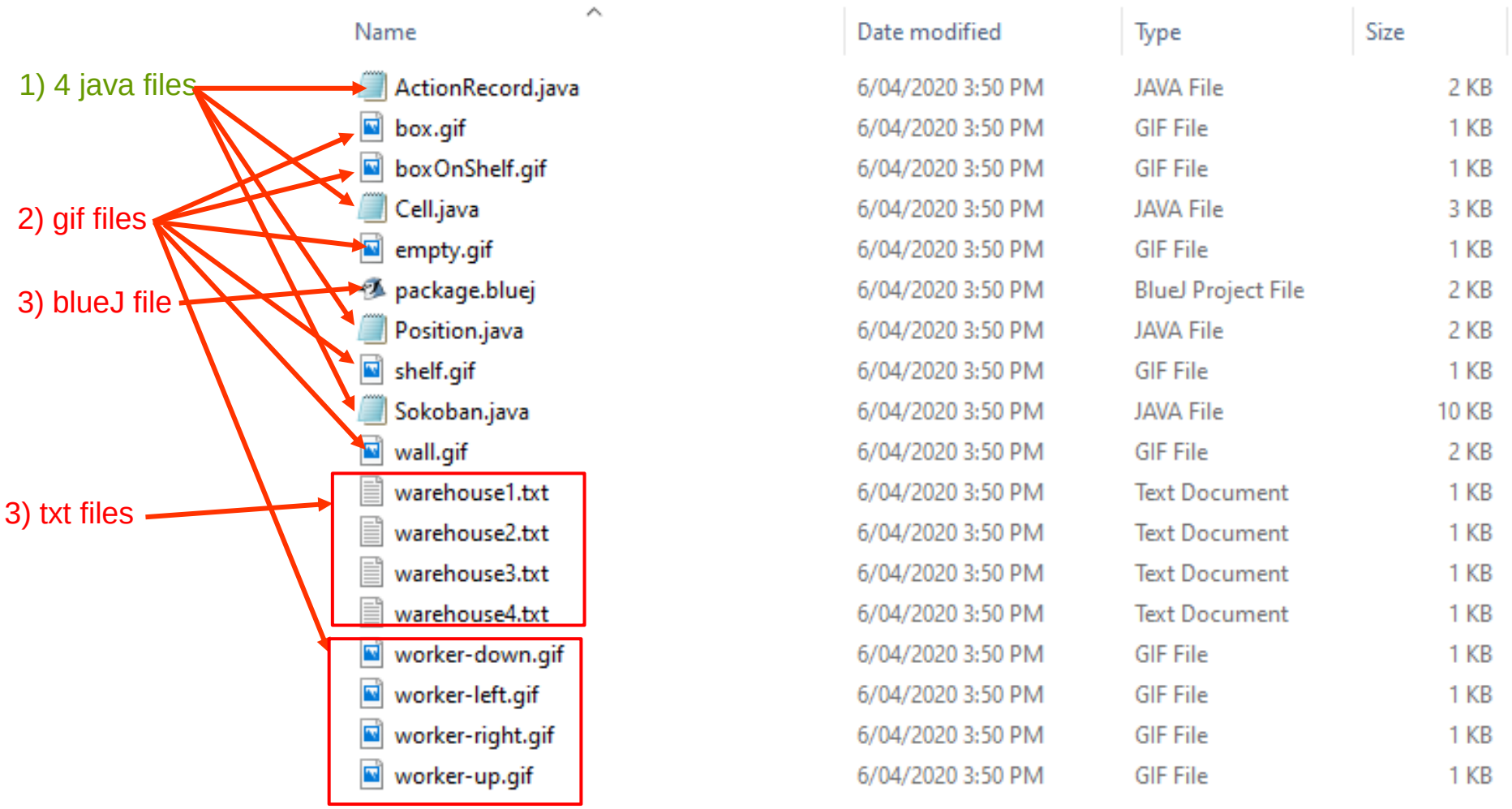
To help you with implementing the undo functionality, we have provided an `ActionRecord` class for recording information about actions.

To implement the undo facility, you need

1. a `history` field that contains a `Stack` of `ActionRecord`
2. extra lines of code in the methods that perform the actions to create an `ActionRecord` object and push it on the history stack.
3. extra lines of code in the `load` method to reset the history stack when the level is reset.
4. an "Undo" button (plus optionally a key mapping from "u" to the undo method).
5. an `undo` method that is called when the "Undo" button is clicked, which pops the top ActionRecord off the stack, and reverses the action.

Core

Files in the **Sokoban folder**



Name	Date modified	Type	Size
ActionRecord.java	6/04/2020 3:50 PM	JAVA File	2 KB
box.gif	6/04/2020 3:50 PM	GIF File	1 KB
boxOnShelf.gif	6/04/2020 3:50 PM	GIF File	1 KB
Cell.java	6/04/2020 3:50 PM	JAVA File	3 KB
empty.gif	6/04/2020 3:50 PM	GIF File	1 KB
package.bluej	6/04/2020 3:50 PM	BlueJ Project File	2 KB
Position.java	6/04/2020 3:50 PM	JAVA File	2 KB
shelf.gif	6/04/2020 3:50 PM	GIF File	1 KB
Sokoban.java	6/04/2020 3:50 PM	JAVA File	10 KB
wall.gif	6/04/2020 3:50 PM	GIF File	2 KB
warehouse1.txt	6/04/2020 3:50 PM	Text Document	1 KB
warehouse2.txt	6/04/2020 3:50 PM	Text Document	1 KB
warehouse3.txt	6/04/2020 3:50 PM	Text Document	1 KB
warehouse4.txt	6/04/2020 3:50 PM	Text Document	1 KB
worker-down.gif	6/04/2020 3:50 PM	GIF File	1 KB
worker-left.gif	6/04/2020 3:50 PM	GIF File	1 KB
worker-right.gif	6/04/2020 3:50 PM	GIF File	1 KB
worker-up.gif	6/04/2020 3:50 PM	GIF File	1 KB

1) ActionRecord.java

ActionRecord.java - Notepad

File Edit Format View Help

1

```
/**
 * Object containing the record of an action (move or push) in
 * a given direction.
 * Used for the Undo process.
 * Every move or push should put an ActionRecord on the history stack
 * Undo should pop an ActionRecord off the history stack and
 *  undo the recorded action.
 */

public class ActionRecord {
    private final boolean isPush;   // if it is not a "push", it is a "move"
    private final String direction; // direction of the move or push

    /**
     * Constructor
     */
    public ActionRecord(String action, String dir) {
        isPush = (action.equalsIgnoreCase("push"));
        direction = dir;
    }

    /**
     * Is the recorded action a push?
     */
    public boolean isPush() {
        return isPush;
    }

    /**
     * Is the recorded action a move?
     */
    public boolean isMove() {
        return !isPush;
    }

    /**
```

2

3

4

ActionRecord.java (continued)

```
    /**
     * Return the direction of the recorded action
     */
    public String direction() {
        return direction;
    }

    /**
     * Return a String describing the recorded action
     */
    public String toString() {
        return ((isPush ? "Push" : "Move") + " to " + direction);
    }

}
```

5

6

2) Cell.java

Cell.java - Notepad

File Edit Format View Help

1. Comments

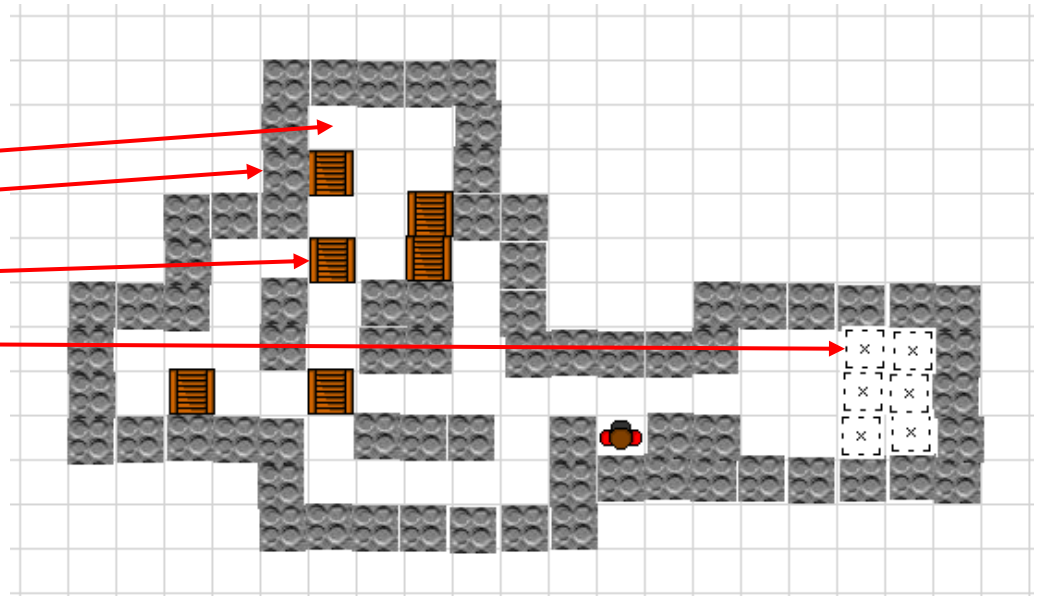
```
// This program is copyright VUW.
// You are granted permission to use it to construct your answer to a XMUT103 assignment
// You may not distribute it in any other way without permission.

/* Code for XMUT103 - 2020T1, Assignment 1
 * Name:
 * Username:
 * ID:
 */

import ecs100.*;

/**
 * A Cell is a single square in a Sokoban game.
 * It can be one of five things:
 *  an "empty" cell
 *  a "wall"
 *  a cell with a "box"
 *  a cell with a "shelf"
 *  a cell with a "box on a shelf"
 * It has several useful methods
 */
```

2



3a

```
public class Cell {
    private String type;

    /**
     * Constructor. The type must be one of the valid types of cells.
     */
    public Cell(String t){
        if (t.equals("empty") || t.equals("shelf") || t.equals("wall") ||
        t.equals("box") || t.equals("boxOnShelf")){
            type = t;
        }
        else {
```

Cell.java (continued) 3b

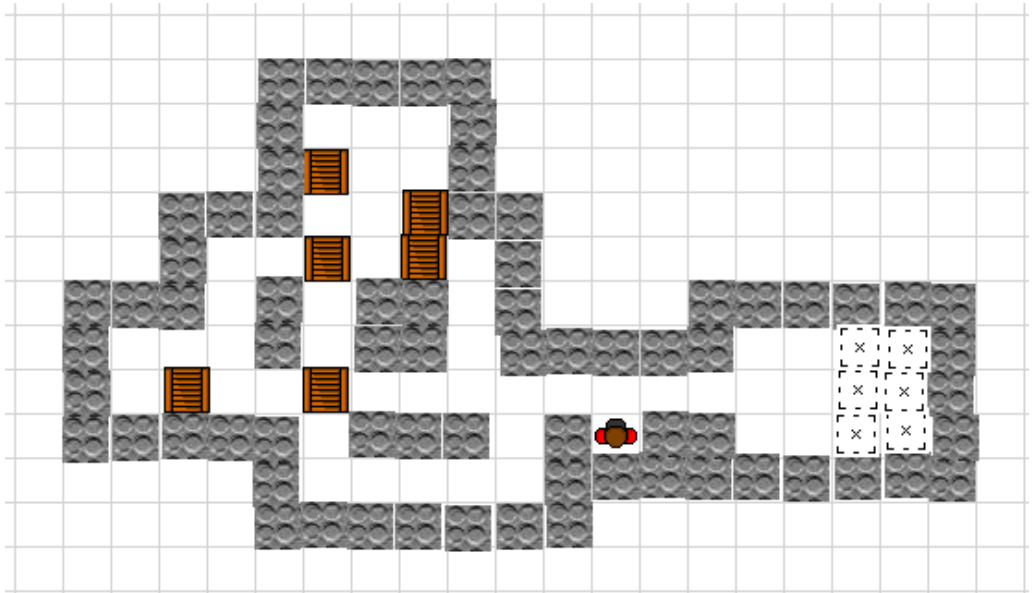
```
    }
    else {
        throw new RuntimeException("Invalid Cell type!");
    }
}
```

4
```
/**
 * Is this cell a shelf without a box?
 */
public boolean isEmptyShelf() {
    return type.equals("shelf");
}
```

5
```
/**
 * Does this cell have a box in it?
 */
public boolean hasBox() {
    return (type.contains("box"));
}
```

6
```
/**
 * Is the cell is free to move onto (not a wall or a box)
 */
public boolean isFree() {
    return (type.equals("empty") || type.equals("shelf"));
}
```

7
```
/**
 * Change the type of the cell to not have a box in it
 */
public void removeBox() {
    if (type.equals("box")) { type = "empty"; }
    else if (type.equals("boxOnShelf")) { type = "shelf"; }
}
```



2) Cell.java (continued)

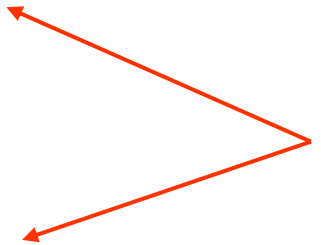
6
```
/**
 * Is the cell is free to move onto (not a wall or a box)
 */
public boolean isFree() {
    return (type.equals("empty") || type.equals("shelf"));
}
```

7
```
/**
 * Change the type of the cell to not have a box in it
 */
public void removeBox() {
    if (type.equals("box")) { type = "empty"; }
    else if (type.equals("boxOnShelf")) { type = "shelf"; }
}
```

8
```
/**
 * Change the type of the cell to contain a box
 */
public void addBox() {
    if (type.equals("empty")) { type = "box"; }
    else if (type.equals("shelf")) { type = "boxOnShelf"; }
}
```

9
```
/**
 * Draw the cell at the specified position and size
 */
public void draw(double left, double top, double size) {
    UI.drawImage((type +".gif"), left, top, size, size);
}
```

```
}
```



6 & 7 are same methods as shown in previous slide

3) Position.java

Position.java - Notepad

File Edit Format View Help

1. Comments

```
// This program is copyright VUW.
// You are granted permission to use it to construct your answer to a XMUT103 assignment.
// You may not distribute it in any other way without permission.

/* Code for XMUT103 - 2020T1, Assignment 1
 * Name:
 * Username:
 * ID:
 */
```

2

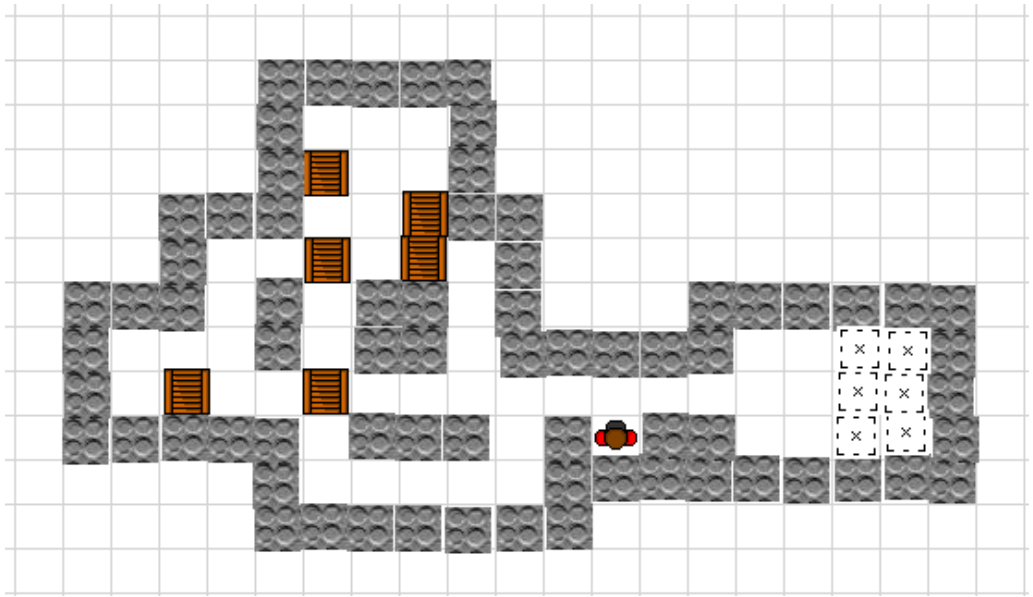
```
/**
 *  A pair of row and column representing the coordinates of a cell in the warehouse.
 *  Has a method to return the next Position in a given direction.
 *  Because the fields are final (can't be changed), it is safe to make
 *  the fields public.
 *  If  pos is a variable containing a Position, then pos.row and pos.col
will be the values of the row and the col in the Position.
 */
```

3

```
public class Position {

    /**
     * Fields containing a row and a column
     */
    public final int row;
    public final int col;

    /**
     * Constructor
     */
    Position (int row, int col) {
        this.row = row;
        this.col = col;
    }
```



3) Position.java

Position.java - Notepad

File Edit Format View Help

1. Comments

```
// This program is copyright VUW.
// You are granted permission to use it to construct your answer to a XMUT103 assignment.
// You may not distribute it in any other way without permission.

/* Code for XMUT103 - 2020T1, Assignment 1
 * Name:
 * Username:
 * ID:
 */
```

2

```
/**
 *  A pair of row and column representing the coordinates of a cell in the warehouse.
 *  Has a method to return the next Position in a given direction.
 *  Because the fields are final (can't be changed), it is safe to make
 *  the fields public.
 *  If  pos is a variable containing a Position, then pos.row and pos.col
will be the values of the row and the col in the Position.
 */
```

3

```
public class Position {

    /**
     * Fields containing a row and a column
     */
    public final int row;
    public final int col;

    /**
     * Constructor
     */
    Position (int row, int col) {
        this.row = row;
        this.col = col;
    }
```

Position.java (continued)

```
    /**
     * Return the next position in the specified direction
     */
    public Position next(String direction) {
        if (direction.equals("up"))    return new Position(row-1, col);
        if (direction.equals("down"))  return new Position(row+1, col);
        if (direction.equals("left"))  return new Position(row, col-1);
        if (direction.equals("right")) return new Position(row, col+1);
        return this;
    }

    /**
     * Return a string with the values of the fields.
     */
    public String toString() {
        return String.format("(%d,%d)", row, col);
    }
}
```