P2: Exercise 1 Discussion

Claudio Corrodi

2 March 2018
Two approaches

- Custom algorithm
  - Recursive
  - Look at the first character of pattern and filename at a time

- Regular expressions
  - One-liner can cover most cases
  - But: What about special character?
private boolean match_rec(String pattern, String filename) {
    ...  
    if (pattern.charAt(0) != filename.charAt(0)) {
        return false;
    } else {
        return match_rec(pattern.substring(1), filename.substring(1));
    }  
    ...
}
Custom Algorithm using recursion

```java
private boolean match_rec(String pattern, String filename) {
    ...
    if (pattern.charAt(0) != filename.charAt(0)) {
        return false;
    } else {
        return match_rec(pattern.substring(1), filename.substring(1));
    }
    ...
}
```

```
match_rec("abc", "abcde.txt") ==
match_rec("bc", "bcde.txt") ==
match_rec("c", "cde.txt") ==
match_rec("", "de.txt") == ...
```
private boolean match_rec(String pattern, String filename) {
    ...
    // Question mark. If filename is not empty, match the remainder
    // of pattern to the remainder of filename.
    if (pattern.startsWith("?")) {
        if (filename.isEmpty()) {
            return false;
        } else {
            return match_rec(pattern.substring(1),
                             filename.substring(1));
        }
    }
}
Custom Algorithm using recursion

```java
private boolean match_rec(String pattern, String filename) {
    ...
    // Question mark. If filename is not empty, match the remainder
    // of pattern to the remainder of filename.
    if (pattern.startsWith("?")) {
        if (filename.isEmpty()) {
            return false;
        } else {
            return match_rec(pattern.substring(1),
                              filename.substring(1));
        }
    }
}
```

`match_rec("?oo.txt", "foo.txt") ==
match_rec("oo.txt", "oo.txt") == ...`
private boolean match_rec(String pattern, String filename) {
    ...
    // Star. Try to match any remainder.
    for (int i = 0; i <= filename.length(); i++) {
        if (match_rec(pattern.substring(1), filename.substring(i))) {
            return true;
        }
    }
    return false;
}
**Custom Algorithm using recursion**

```java
private boolean match_rec(String pattern, String filename) {
    ...
    // Star. Try to match any remainder.
    for (int i = 0; i <= filename.length(); i++) {
        if (match_rec(pattern.substring(1), filename.substring(i))) {
            return true;
        }
    }
    return false;
}
```

- `match_rec("*.txt", "foo.txt")` ==
- `match_rec(".txt", "foo.txt")` ∨
- `match_rec(".txt", "oo.txt")` ∨
- `match_rec(".txt", "o.txt")` ∨
- `match_rec(".txt", "txt")`
private boolean matchRegex(String filename) {
    String regexPattern = pattern;
    regexPattern = regexPattern.replace("*", ".*");
    regexPattern = regexPattern.replace(\?, ".");
    return Pattern.matches(regexPattern, filename);
}
private boolean matchRegex(String filename) {
    String regexPattern = pattern;
    regexPattern = regexPattern.replace("*", ".*" methodology);
    regexPattern = regexPattern.replace("?", ".");
    return Pattern.matches(regexPattern, filename);
}

"." matches exactly one character
".*" matches any number of characters

• What about special characters?
  ⇒ Read the documentation!

    regexPattern = regexPattern.replace(".", "\\.");
Examples: Encapsulation & names

```java
public class FilePattern {

    public String string;

    public FilePattern(String string) {
        this.string = string;
    }
}
```
Examples: Encapsulation & names

```java
public class FilePattern {
    public String string;

    public FilePattern(String string) {
        this.string = string;
    }
}
```
Examples: Encapsulation & names

```java
public class FilePattern {
    protected String pattern;
    public FilePattern(String pattern) {
        this.string = string;
    }
}
```

Make attributes protected

Use meaningful names
Examples: Useless code

```java
protected String tempPattern;

public String getTempPattern() {
    return this.tempPattern;
}
```
Examples: Useless code

```java
protected String tempPattern;

public String getTempPattern() {
    return this.tempPattern;
}
```

Unused outside of class! Use tempPattern directly.
public class TestMain {
    public static void main(String[] args) {
        FilePattern a = new FilePattern("fname*");
        System.out.println(a.matches("fname.txt"));
    }
}

public class FilePatternTest {
    @Test
    public void fnameStarMatchesFnameDotTxt() {
        FilePattern a = new FilePattern("fname*");
        assertTrue(a.matches("fname.txt"));
    }
}

Add the scenario as a permanent test
Manual Testing

```java
public class TestMain {
    public static void main(String[] args) {
        FilePattern a = new FilePattern("fname*");
        System.out.println(a.matches("fname.txt"));
    }
}
```

```java
public class FilePatternTest {
    @Test
    public void fnameStarMatchesFnameDotTxt() {
        FilePattern a = new FilePattern("fname*");
        assertTrue(a.matches("fname.txt"));
    }
}
```

Add the scenario as a permanent test
Javadoc

Alex Syrel

P2
Java supports three comment types:

```java
/**
 * A documentation comment.
 */

/*
 * A standard comment.
 */

// A one-line comment.
```
Java supports three comment types:

```java
/**
 * A documentation comment.
 */

/*
 * A standard comment.
 */

// A one-line comment.
```
Why to document?
Code is read much more often than it is written
Even if you don't intend anybody else to read your code, that somebody is probably going to be you, twelve months from now.
public Affine2 setToTrnRotScl(float x, float y, float degrees, float scaleX, float scaleY) {
    m02 = x;
    m12 = y;

    if (degrees == 0) {
        m00 = scaleX;
        m01 = 0;
        m10 = 0;
        m11 = scaleY;
    } else {
        float sin = MathUtils.sinDeg(degrees);
        float cos = MathUtils.cosDeg(degrees);

        m00 = cos * scaleX;
        m01 = -sin * scaleY;
        m10 = sin * scaleX;
        m11 = cos * scaleY;
    }

    return this;
}

https://github.com/libgdx/libgdx/blob/master/gdx/src/com/badlogic/gdx/math/Affine2.java
Inform others how to use your code without having to read it

```java
public Affine2 setToTrnRotScl(
    m02 = x;
    m12 = y;
    if (degrees == 0) {
        m00 = scaleX;
        m01 = 0;
        m10 = 0;
        m11 = scaleY;
    } else
    float
    float
    m00 = cos * scaleX;
    m01 = -sin * scaleY;
    m10 = sin * scaleX;
    m11 = cos * scaleY;
} return
```

/** Sets this matrix to a concatenation of translation, rotation and scale. * It is a more efficient form for: * <code>idt().translate(x, y).rotate(degrees).scale(scaleX, scaleY)</code> * @param x The translation in x. * @param y The translation in y. * @param degrees The angle in degrees. * @param scaleX The scale in y. * @param scaleY The scale in x. * @return This matrix for the purpose of chaining operations. */

https://github.com/libgdx/libgdx/blob/master/gdx/src/com/badlogic/gdx/math/Affine2.java
What is Good Documentation?
Make the first sentence count

Javadoc assumes it to be the summary

/**
 * When I was a kid I had absolutely no idea
 * the day will come when I stop writing code
 * and begin to do JavaDoc.
 * Nevertheless this method returns 42.
 *
 * @return 42
 */
Do not use fillers!

This method/function/class... is not necessary.

/**
 * This is a nice method to assert beautiful quality
 * of amazing chars at a given index under the moonlight
 */
First word should be a verb

helps to understand code faster

/**
 * Removes user from the list
 */

/**
 * Translates window to the left
 */

/**
 * Establishes network connection
 */
Remember to describe corner cases. e.g. null? negative ints?

/**
 * ...
 * Moves snake to specified position.
 * Snake should not be null as long as position is positive and less then 10
 * ...
 */

public void moveTo(int position) { }
Would be nice

link to other documentation – with @see or @link

/**
 * Convenience for calling {@link Window#getLayoutInflater}.
 *
 * @see android.view.Window
 */

public LayoutInflater getLayoutInflater() {
    return getWindow().getLayoutInflater();
}
Class Comments
What is the class responsible for?
What information does it hold?
What things can it do?
Who uses this class?
How should the class be used?
Does this class need special treatment?
### Activity Lifecycle

An activity is a single, focused thing that the user can do. Almost all activities interact with the user through a single window, and this window is the focus of the user’s attention. Many activities have a life cycle that begins with the user requesting that an activity be run and ends with the activity being destroyed. The following diagram shows the important state paths of an Activity.

<table>
<thead>
<tr>
<th>State Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created</td>
<td>The activity is created as an instance of the activity class.</td>
</tr>
<tr>
<td>Started</td>
<td>The activity is started as a result of activity stack management or an intent.</td>
</tr>
<tr>
<td>Resumed</td>
<td>The activity is resumed because it was paused and then resumed.</td>
</tr>
<tr>
<td>Destroyed</td>
<td>The activity is destroyed because it is being stopped.</td>
</tr>
</tbody>
</table>

Activities in the system are managed as an activity stack, which can be thought of as a list of all activities that are currently running. When an activity is started, it is placed on the top of the activity stack, with user input going to it. An activity will still be restarted and resumed if it is killed by the system, so its lifecycle is not killed by the system. However, if the activity is killed by the system, it will be destroyed, and the system will call onDestroy() to the activity. The following table shows the important state paths of an Activity, along with the methods that are called in each state.

<table>
<thead>
<tr>
<th>State Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created</td>
<td>The activity is created as an instance of the activity class.</td>
</tr>
<tr>
<td>Started</td>
<td>The activity is started by the activity stack manager.</td>
</tr>
<tr>
<td>Resumed</td>
<td>The activity is resumed because it was paused and then resumed.</td>
</tr>
<tr>
<td>Destroyed</td>
<td>The activity is destroyed because it is being stopped.</td>
</tr>
</tbody>
</table>

### Activity Creation

An activity is created as an instance of the activity class. The following method is called when the activity is first created.

```java
protected void onCreate(Bundle savedInstanceState) { 
    // code to initialize the activity
}
```

### Activity Start

An activity can be started in a variety of ways, including:
- Starting with an explicit intent
- Starting with a startActivity() call
- Starting with an implicit intent
- Starting with an activity stack manager

### Activity Resume

An activity is resumed when it is moved to the foreground, or
- when the activity receives input
- when the activity finishes
- when the activity is restarted

### Activity Paused

An activity is paused when it is removed from the top of the activity stack, causing it to lose the user’s focus. The following method is called when the activity is paused.

```java
protected void onPause() { 
    // code to stop work when the activity is lost focus
}
```

### Activity Resume

An activity is resumed when it is moved to the foreground, or
- when the activity receives input
- when the activity finishes
- when the activity is restarted

### Activity Destroyed

An activity is destroyed when it is being stopped. The following method is called when the activity is destroyed.

```java
protected void onDestroy() { 
    // code to destroy the activity
}
```

### Activity Configuration Changed

Activities can change based on any configuration value. Thus the activity’s lifecycle includes a configuration change to handle an activity that is changing configuration.

```java
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    // code to handle configuration change
}
```

### Activity LifeCycle

The following diagram shows the important state paths of an Activity.

- **Created**: The activity is created as an instance of the activity class.
- **Started**: The activity is started as a result of activity stack management or an intent.
- **Resumed**: The activity is resumed because it was paused and then resumed.
- **Destroyed**: The activity is destroyed because it is being stopped.

### Activity Stack Management

Activity stack management is responsible for managing the activities that are running in the system. It is the responsibility of the activity stack manager to:
- Start activities
- Resume activities
- Pause activities
- Destroy activities

### Activity Stack

Activities in the system are managed as an activity stack. An activity stack is a list of all activities that are currently running. When an activity is started, it is placed on the top of the activity stack, with user input going to it. An activity will still be restarted and resumed if it is killed by the system, so its lifecycle is not killed by the system. However, if the activity is killed by the system, it will be destroyed, and the system will call onDestroy() to the activity.
Method Comments
Remember to describe

Parameters (@param)

/**
 * Throws an appropriate exception based on the passed in error code.
 *  
 * @param code - the DND error code, should be positive
 */

public static void error (int code) {
    error (code, 0);
}

/**
 * Get the source of this exception event.
 *
 * @return The {@link Throwable} that is the source of this exception event.
 */

public Throwable getException() {
    return (Throwable) getSource();
}
Remember to describe

Exceptions (@throws)

/**
 * ...
 * @throws android.content.ActivityNotFoundException
 * if there was no Activity found to run the given Intent.
 * ...
 */

public void startActivityForResult(Intent intent, int requestCode)
    throws ActivityNotFoundException {
    startActivityForResult(intent, requestCode, null);
}

https://github.com/android/platform_frameworks_base/blob/master/core/java/android/app/Activity.java
Examples
public class ServerProxy implements IServer {
    /* ... */
}

/**
 * Relays method calls to a remote { @see Server }.<p>
 * The proxy is responsible for establishing and keeping a connection to the server. The caller must ensure that a connection is destroyed with the {@see #disconnect} method.
 */

public class ServerProxy implements IServer {
    /* ... */
}
public ServerProxy(String url, int port) throws NetworkConnectionException {
/* ... */
}
/**
 * Constructor
 */
public ServerProxy(String url, int port)
throws NetworkConnectionException {
/* ... */
}
public ServerProxy(String url, int port)
throws NetworkConnectionException {
/* ... */
}
/**
 * Establishes a connection to a remote server.
 * Throws if it fails to do so.
 *
 * @param url address that can either be resolved via hosts.conf or DNS or is an IP address.
 *
 * @param port port to connect to on the server. A positive integer, typically above 1024.
 * Must be the same as the {@see Server} uses with its {@see Server#listenOn} method.
 *
 * @throws NetworkConnectionException if it was not able to initiate a connection.
 */
public ServerProxy(String url, int port)
    throws NetworkConnectionException {
    /* ... */
}
/**
 * Ends the connection
 */
public void disconnect () {
   // ...
}
/**
 * Returns the number of jobs
 */
public int getJobCount () {
   // ...
}
/**
 * Ends the connection. After this call, no other
 * method call is valid, including this one. The
 * server is not affected by this.
 */

public void disconnect () {
    // ...
}

/**
 * Returns the number of jobs running on the server.
 * @return a non-negative integer that is the number of jobs that are alive.
 */
public int getJobCount () {
    // ...
}
/**
 * Returns the url of the server.
 */

public String getUrl () {
    return url;
}
public String getUrl () {
    return url;
}
Sometimes no comments are best comments
The end
Exercise 2
Exercise 2: Snakes & Ladders

- You are given a skeleton for the Snakes & Ladders game
- Add new types of squares
- Test behaviour of squares (using JUnit)
- Write proper documentation
JUnit

- Testing framework
  - Covered in more detail in lecture 4

- Goal: Make sure program behaves as expected

- JUnit: Individual, independent tests
@Test
public void newGame() {
    jack = new Player("Jack");
    jill = new Player("Jill");
    Player[] args = { jack, jill };
    Game game = new Game(12, args);
    game.setSquareToLadder(2, 4);
    game.setSquareToLadder(7, 2);
    game.setSquareToSnake(11, -6);
    assertTrue(game.notOver());
    assertTrue(game.firstSquare().isOccupied());
    assertEquals(1, jack.position());
}
@Test
public void newGame() {
    jack = new Player("Jack");
    jill = new Player("Jill");
    Player[] args = { jack, jill };  
    Game game = new Game(12, args);
    game.setSquareToLadder(2, 4);
    game.setSquareToLadder(7, 2);
    game.setSquareToSnake(11, -6);
    assertTrue(game.notOver());
    assertTrue(game.firstSquare().isOccupied());
    assertEquals(1, jack.position());
}
@Test
public void newGame() {
    jack = new Player("Jack");
    jill = new Player("Jill");
    Player[] args = { jack, jill };  
    Game game = new Game(12, args);
    game.setSquareToLadder(2, 4);
    game.setSquareToLadder(7, 2);
    game.setSquareToSnake(11, -6);
    assertTrue(game.notOver());
    assertTrue(game.firstSquare().isOccupied());
    assertEquals(1, jack.position());
}
@Test
public void newGame() {
    jack = new Player("Jack");
    jill = new Player("Jill");
    Player[] args = { jack, jill };  
    Game game = new Game(12, args);
    game.setSquareToLadder(2, 4);
    game.setSquareToLadder(7, 2);
    game.setSquareToSnake(11, -6);
    assertTrue(game.notOver());
    assertTrue(game.firstSquare().isOccupied());
    assertEquals(1, jack.position());
}

The exercise comes with some existing tests for reference.

More in exercise_02.md
git pull p2-exercises master