P2: Exercise 1 Discussion

Pooja Rani
1 March 2019
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?
Custom Algorithm using recursion

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") ==
Regular expressions

private boolean matchRegex(String filename) {
    String regexPattern = pattern;
    regexPattern = regexPattern.replace( "\*", "\.*" );
    regexPattern = regexPattern.replace( "\?", "\." );
    return Pattern.matches( regexPattern, filename );
}

• What about special characters? ⇒ Read the documentation!

"." matches exactly one character
".*" matches any number of characters
public class FilePattern{

    public String string;

    public FilePattern(String string) {
        this.string = string;
    }

}
public class FilePattern{

    public String string;

    public FilePattern(String string ) {
        this.string = string;
    }
}

Examples: Encapsulation & names
public class FilePattern {
    protected String string;

    public FilePattern(String string) {
        this.string = string;
    }
}

Examples: Encapsulation & names

Make attributes protected
public class FilePattern {
    protected String pattern;

    public FilePattern(String pattern) {
        this.pattern = pattern;
    }
}

Examples: Encapsulation & names

- 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 TestMain {
    public static void main(String[] args) {
        FilePattern a = new FilePattern("fname*");
        System.out.println(a.matches("fname.txt"));
    }
}
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"));
    }
}
public class FilePatternTest {
    @Test
    public void fnameStarMatchesFnameDotTxt() {
        FilePattern a = new FilePattern("fname*");
        assertTrue(a.matches("fname.txt"));
    }
}
Javadoc
**Javadoc**

**Javadoc**: Program to generate java code documentation.

**Input**: Java source file (.java)

**Output**: HTML files documenting specification of java code.
Comment types

/**
 * A documentation comment
 */

/**
 * A standard comment
 */

// One-line comment
Why to document?
Why to document?

*Code is read much more often than it is written*
Why to document?

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;
}
/** 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. */

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;
}
/** Sets this matrix to a concatenation of translation, rotation and scale. It is a more efficient form for:
 * \[ \text{idt().translate(x, y).rotate(degrees).scale(scaleX, scaleY)} \]
 * @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
/** 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. */
/** 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. */
What is Good Documentation?
What is Good Documentation?

/**
 * 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
 */
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

Javadoc assumes first lines to be the summary.
What is Good Documentation?

/**
 * This is a nice method to assert beautiful quality
 * of amazing chars at a given index under the moonlight
 */
What is Good Documentation?

/**
 * This is a nice method to assert beautiful quality
 * of amazing chars at a given index under the moonlight
 */
What is Good Documentation?

Do not use fillers

This method/function/class.. is not necessary.
What is Good Documentation?

First word should be a verb
helps to understand code faster

/**
 * Removes user from the list
 */
/**
 * Translates window to the left
 */
/**
 * Establishes network connection
 */
What is Good Documentation?

Remember to describe corner cases
e.g. null? negative ints?

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

public void moveTo(int position) { }
```
What is Good Documentation?

Link to other documentation with @see or @link

/**
 * Returns result of {@link #matchesFilenameAndPattern(String, String)}.
 * Test methods like *
 * {@link FilePatternTest#fnameStarShouldNotMatch()}
 * calls this method.
 * @param filename filename to compare
 * @return true if filename matches the pattern
 */

public boolean matches(String filename) {
    return this.matchesFilenameAndPattern(filename, "a?.text" );
}
Class Comments
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?
/**
 * Filters file names using command-line wildcards.
 * 
 * '*' matches any number of characters.
 * '?' matches exactly one character.
 * 
 * Examples:
 * '*.md' matches all files with the markdown extension.
 * 'exercise_??_.md' matches, for example, 'exercise_01.md'.
 * 
 * @see FilePatternTest uses this class.
 * @version 1.0.0
 * @author You!
 * */

public class FilePattern {
/**
 * Filters file names using command-line wildcards.
 * *
 * '*' matches any number of characters.
 * '?' matches exactly one character.
 *
 * Examples:
 * '*.md' matches all files with the markdown extension.
 * 'exercise_???.md' matches, for example, 'exercise_01.md'.
 *
 * @see FilePatternTest uses this class.
 * @version 1.0.0
 * @author You!
 */

public class FilePattern {

/**
 * Filters file names using command-line wildcards.
 * 
 * '*' matches any number of characters.
 * '?' matches exactly one character.
 * 
 * Examples:
 * '*.md' matches all files with the markdown extension.
 * 'exercise_??.md' matches, for example, 'exercise_01.md'.
 * 
 * @see FilePatternTest uses this class.
 * @version 1.0.0
 * @author You!
 */

public class FilePattern {

/**
 * Filters file names using command-line wildcards.
 * 
 * '\*' matches any number of characters.
 * '\?' matches exactly one character.
 * 
 * Examples:
 * '\*.md' matches all files with the markdown extension.
 * 'exercise_??.md' matches, for example, 'exercise_01.md'.
 * 
 * @see FilePatternTest uses this class.
 * @version 1.0.0
 * @author You!
 */

public class FilePattern {
/**
 * Filters file names using command-line wildcards.
 * 
 * '*' matches any number of characters.
 * '?' matches exactly one character.
 * 
 * Examples:
 * '*.md' matches all files with the markdown extension.
 * 'exercise_???.md' matches, for example, 'exercise_01.md'.
 * 
 * @see FilePatternTest uses this class.
 * @version 1.0.0
 * @author You!
 */

public class FilePattern {

Method Comments
/** 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. */

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;
}
/** 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. */

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);
        m11 = cos * scaleX;
    }
    return this;
}
public void startActivityForResult(Intent intent, int requestCode)
throws ActivityNotFoundException {
    startActivityForResult(intent, requestCode, null);
}

@throws Exception
https://github.com/android/platform_frameworks_base/blob/master/core/java/android/app/Activity.java
public class FilePattern {

    /**
     * Creates a new instance of the FilePattern class that filters
     * file names based on the given pattern
     * @param pattern the pattern used to filter file names.
     * @see FilePattern
     */

    public FilePattern(String pattern) {
        // your implementation
    }
}
Tags

Package Tags

- `@see`
- `@since`
- `@author`
- `@version`
- `{@link}`
- `{@docRoot}`
Tags

Class/Interface Tags

• @see
• @since
• @deprecated
• @author
• @version
• {@link}
• {@docRoot}
Tags

Field Tags
• @see
• @since
• @deprecated
• {@value}
• {@link}
• {@docRoot}
Method/Constructor Tags

- @see
- @since
- @deprecated
- @param
- @return
- @throws / @exception
- {@link}
- {@docRoot}
Tag names are case-sensitive. @See is a mistaken usage. @see is correct.
Sometimes no comments are best comments
/**
 * The end
 */
P2: Exercise 2
Exercise 2: Snakes & Ladders

- You are given a skeleton for the Snakes and 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
class TestGame {
    @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());
}

initialise player
@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());
}

Specify expected output
```java
@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());
equal(1, jack.position());
}
```