P2: Exercise 2 Discussion

Claudio Corrodi

Exercise 2

Wormhole: How to find all the exits?

- Main problem: Entrances need to be aware of all exits!

Approaches

- Let the Game keep track of exits
- Static list in WormholeExit class. Add “this” to the list when constructing an exit.
- (… more advanced / dynamic solutions possible …)

Exercise 2

SwapSquare: In the “landHereOrGoHome()” method, how do we move both players?

- No need for the active player!
- But we need to instruct the other player to move:

```
@Override
public ISquare landHereOrGoHome() {
    int otherPosition = game.nextPlayer().position();
    game.nextPlayer().goToSquare(this);
    return game.getSquare(otherPosition);
}
```

Exercise 2

SwapSquare: In the “landHereOrGoHome()” method, how do we move both players?

- No need for the acti
- But we need to instruct the other play ove:

Find the target position for the active player

```
@Override
public ISquare landHereOrGoHome() {
    int otherPosition = game.nextPlayer().position();
    game.nextPlayer().goToSquare(this);
    return game.getSquare(otherPosition);
}
```

Exercise 2

Swap S … e()”
metho …

- No r …
- But … move:

```
@Override
public ISquare la…  eOrGoHome() {
    int otherPosition …  game.nextPlayer().position();
    game.nextPlayer().goToSquare(this);
    return game.getSquare(otherPosition);
}
```

Instruct other player to change his square to “this” (i.e. the SwapSquare)

```
public ISquare goToSquare(ISquare newSquare) {
    ISquare oldSquare = this.square();
    oldSquare.leave(this);
    newSquare.enter(this);
    square = newSquare;
    return oldSquare;
}
```

Exercise 2

SwapSquare: In the “landHereOrGoHome()” method, how do we move both players?

- Need for the active player!
- Ask the other player to move:

Return the old square of the other player

```
@Ove
publ    Square landHereOrGoHome() {
    in  otherPosition = game.nextPlayer().position();
    gam e.nextPlayer().goToSquare(this);
    return game.getSquare(otherPosition);
}
```

Exercise 2

SwapSquare: In the “landHereOrGoHome()” method, how do we move both players?

- No nee
- But we r to move:

Delegation of responsibility:
We care about our new square, but we don't do the other player's job!

```
@Override
public ISquare landHereOrGoHome() {
    int otherPosition = game.nextPlayer().position();
    game.nextPlayer().goToSquare(this);
    return game.getSquare(otherPosition);
}
```

P2: Design by Contract, Assertions & Exceptions

Claudio Corrodi

Exception or Assertion?

```
/**
 * Sets the refresh rate for the current display.
 * @param rate
 */
public void setRefreshRate(int rate) {
   // what if rate < 0?
}
```

Exception or Assertion?

```
/**
 * Sets the refresh rate for the current display.
 * @param rate new refresh rate, must be >= 0
 */
public void setRefreshRatePrecondition(int rate) {
   assert rate >= 0;
}
```

Exception or Assertion?

```
/**
 * Sets the refresh rate for the current display.
 * @param rate new refresh rate
 * @throws IllegalArgumentException if rate is not valid
 */
public void setRefreshRateException(int rate)
      throws IllegalArgumentException {
   if (rate < 0) {
      throw new IllegalArgumentException();
   }
}
```

Assertions

- Use when you expect a property to hold
- Use for contracts
  - Pre-/postconditions, invariants
- Use inside complex code
  - E.g. in an algorithm to make sure an intermediate result holds

Assertions

```
/**
 * Draw a vertical line, starting from position,
 * with a length of steps + 1.
 *
 * @param position start location of the line, must not be null
 * @param steps length of the line
 */
public void drawVertical(Point position, int steps) {
    assert position != null;

    // Implementation omitted

    assert(invariant());
}
```

Assertions

- Favor assertions/preconditions for checking method parameters in private/internal API
  - Senders come from within your project → go fix the bug!
  - Simplifies design
- Use assertions for postconditions and invariants

Exceptions

- Error handling
- Expected behaviour
  - Deal with it in try-catch blocks, or
  - Throw it up to the caller

```
public void foobar() throws TodoException {
    throw new TodoException();
}
```

Exceptions

- Do not abuse exceptions

```
try {
   int index = 0;
   while (true) {
      players[index++] = new Player();
   }
} catch (ArrayIndexOutOfBoundsException e) {}
```

Exceptions

- Do not abuse exceptions

```
for (int index = 0;
     index < players.length;
     index++) {
   players[index] = new Player();
}
```

Exceptions

- Favor exceptions for checking method parameters in public/external API
  - Can't trust user to read JavaDoc
- Always use exceptions to check user input!

Checked and Unchecked Exceptions

- Checked exceptions must either be declared

```
public void foobar() throws TodoException {/*...*/}
```

- Or wrapped inside a try-catch block

```
public void bar() {
   try {
       // something that throws a TodoException
   } catch (TodoException e) {
       // handle exception
   }
}
```

- Use checked exceptions unless you have a very good reason not to!

NullPointerException

- Very common unchecked exception
- Often hard to tell where it came from
  - Value may be passed around for a while before it is used
- Include null checks where appropriate

NullPointerException

```
private void newGame() {
   setPlayer(null);
   execute();
}

private void setPlayer(Player player) {
   this.player = player;
}

private void execute() {
   this.player.move();
}
```

NullPointerException

```
Exception in thread "main" java.lang.NullPointerException

    at exercise_03.SomeClass.execute(SomeClass.java:79)
    at exercise_03.SomeClass.newGame(SomeClass.java:65)
    at exercise_03.SomeClass.main(SomeClass.java:7)
    …

Process finished with exit code 1
```

```
private
    setP
    exec
}

private
    this.player = player;
}

private void execute() {
    this.player.move();
}
```

NullPointerException

```
Exception in thread "main" java.lang.NullPointerException

    at exercise_03.SomeClass.execute(SomeClass.java:79)
    at exercise_03.SomeClass.newGame(SomeClass.java:65)
    at exercise_03.SomeClass.main(SomeClass.java:7)
    …

Process finished with exit code 1
```

```
private
    setP
    exec
}

private
    this.player = player;
}

private void execute() {
    this.player.move();
}
```

Why is player == null here?

NullPointerException

```
private void newGame() {
   setPlayer(null);
   execute();
}

/** @param player must not be null */
private void setPlayer(Player player) {
   assert player != null;
   // ...
}

private void execute() {
   this.player.move();
}
```

NullPointerException

```java
private void newGame() {
   setPlayer(null);
   execute();
}

/** @param player must not be null */
private void setPlayer(Player player) {
   assert player != null;
   // ...
}

priv
   p
}
```

```
Exception in thread "main" java.lang.AssertionError

    at exercise_03.SomeClass.setPlayer(SomeClass.java:74)
    at exercise_03.SomeClass.newGame(SomeClass.java:64)
    at exercise_03.SomeClass.main(SomeClass.java:7)

Process finished with exit code 1
```

Another example

```
/**
 * Look up the object at the top of
 * this stack and return it.
 *
 * @return the object at the top
 */
public E top() {
    return top.item;
}
```

Another example

```
/**
 * Look up the object at the top of
 * this stack and return it.
 *
 * @return the object at the top
 */
public E top() {
    return top.item;
}
```

What if the stack is empty?

Another example

```
/**
 * Look up the object at the top of
 * this stack and return it.
 * Returns null if called on an empty stack.
 *
 * @return the object at the top
 */
public E top() {
    if (this.isEmpty())
        return null;
    return top.item;
}
```

Another example

```
/**
 * Look up the object at the top of
 * this stack and return it.
 * Returns null if called on an empty stack.
 *
 * @return the object at the top
 */
public E top() {
    if (this.isEmpty())
        return null;
    return top.item;
}
```

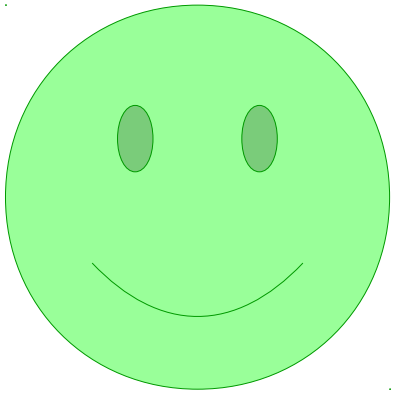
What happens if the stack contains null values?

Another example

```
/**
 * Look up the object at the top of
 * this stack and return it.
 * Throws an EmptyStackException this
 * stack is empty.
 *
 * @return the object at the top
 */
public E top() throws EmptyStackException {
    if (this.isEmpty())
        throw new EmptyStackException();
    return top.item;
}
```

Another example

```
/**
 * Look up the object at the top of
 * this stack and return it.
 * Throws an EmptyStackException this
 * stack is empty.
 *
 * @return the object at the top
 */
public E top() throws EmptyStackException {
    if (this.isEmpty())
        throw new EmptyStackException();
    return top.item;
}
```



P2: Exercise 3

Claudio Corrodi

Turtle Game

- A turtle that moves around a 100x100 board
  - Move up, down, left, right
  - Jump to specific square
  - Leave a red trail

- Input: String representing a turtle program

```
right 10
down 5
jump 20 20
up 5
left 3
```

Turtle Game

Demo

Turtle Game

- You start with
  - TurtleRenderer: GUI
  - BoardMaker: Class that gets text from GUI and returns a Boolean array of size 100x100
- You implement
  - Parse input program (split lines into commands)
  - Execute turtle actions
  - Keep track of trail

Turtle Game

- You start with
  - TurtleRenderer: GUI
  - BoardMaker: Class that gets text from GUI and returns a B
- You
  - P
  - Execute turtle actions
  - Keep track of trail

As always: git pull origin master

Read exercise_03.md