

P2 - Exercise Hours

Iterative Development

27 March, 2020

Pooja Rani

Iterative Development

“The spreadsheet contain cells holding strings”

Model 'cells'

```
public class Cell {  
    private String stringValue;  
  
    public Cell(String stringValue) {  
        this.stringValue = stringValue;  
    }  
  
    public String toString() {  
        if (stringValue != null)  
            return stringValue;  
    }  
}
```

Requirement changes

“The spreadsheet should now contain two different type of cells holding integers or strings”

Model 'cells'

```
public class Cell {
    private String stringValue;
    private int intValue;

    public Cell(String stringValue) {
        this.stringValue = stringValue;
    }

    public Cell(int intValue) {
        this.intValue = intValue;
    }

    public String toString() {
        if (stringValue != null)
            return stringValue;
        else
            return ""+intValue;
    }
}
```

Use OOP

```
public interface Cell {  
    public String toString();  
}
```

```
public class IntegerCell implements Cell {  
    private int value;  
    public IntegerCell(int value) {  
        this.value = value;  
    }
```

```
    public String toString() {  
        return "" + value;  
    }  
}
```

```
public class StringCell implements Cell {  
    private String value;  
    public StringCell(String value) {  
        this.value = value;  
    }
```

```
    public String toString() {  
        return value;  
    }  
}
```

Model 'cells'

```
public class Cell<E> {  
    private E value;  
  
    public Cell(E value) {  
        this.value = value;  
    }  
  
    public String toString() {  
        return this.value.toString();  
    }  
}
```

Generic

```
public class Cell<E> {  
    private E value;  
  
    public Cell(E value) {  
        this.value = value;  
    }  
  
    public String toString() {  
        return this.value.toString();  
    }  
}
```

Generic Type

Generic

```
public class Cell<E> {  
    private E value;  
  
    public Cell(E value) {  
        this.value = value;  
    }  
  
    public String toString() {  
        return this.value.toString();  
    }  
}
```

Generic Type

```
public static void main(String args[]) {  
    Cell<Integer> aCell = new Cell<Integer>(3);  
}
```

Since Java 7

```
public static void main(String args[]) {  
    Cell<Integer> aCell = new Cell(3);  
}
```

Generics

- > Most commonly used type parameters names are:
 - E - Element
 - K - Key
 - N - Number
 - T - Type
 - V - Value

Generics

```
List list = new ArrayList();  
list.add("hello");  
String s = (String) list.get(0);  
}
```

Elimination of casts

```
List<String> list = new ArrayList<String>();  
list.add("hello");  
String s = list.get(0);  
}
```

Stronger Type checks at compile time


'final' classes

What are 'final' classes?

```
final class Bicycle {  
  
}
```

Final classes cannot be extended

```
final class Bicycle {  
  
}  
  
class MountainBike extends Bicycle {  
  
}
```

 The type MountainBike cannot subclass the final class Bicycle

Compilation error

'final' methods

What are 'final' methods?

```
class Bicycle {  
public final void speedUp() {  
  
}  
}
```


*Final methods cannot be overridden
in subclasses*

'final' methods

What are 'final' methods?

```
class Bicycle {  
    public final void speedUp() {  
  
    }  
}
```

```
class MountainBike extends Bicycle {  
    public void speedUp() {  
  
    }  
}
```

 Cannot override the final method from Bicycle

Compilation error

'final' method parameters

What are 'final' method parameters?


```
class Bicycle {  
public void replaceParts(final Part[] parts) {  
  
}  
}
```


Final method parameters cannot be reassigned in the method. (the state of the object can change but not its identity)

'final' method parameters

What are 'final' method parameters?

```
class Bicycle {  
public void replaceParts(final Part[] parts) {  
parts = null;  
parts = new Part[2];
```

 The final local variable parts cannot be assigned. It must be blank and not using a compound assignment

 The final local variable parts cannot be assigned. It must be blank and not using a compound assignment

Compilation error

```
    parts[0] = new Part();  
}  
}
```

OK - The state of the object is changed not its identity

'final' method parameters

What are 'final' method parameters?

```
class Bicycle {  
    public void replaceParts(final Part[] parts) {  
    }  
}  
class MountainBike extends Bicycle {  
    public void replaceParts(Part[] parts) {  
    }  
}
```

The final modifier for method parameters does not influence method overriding

'final' local variables

What are 'final' local variables?

```
class Bicycle {  
public void replaceParts(Part[] parts) {  
final Part[] currentParts = getParts();  
}  
}
```

Final local variables can be assigned only once (the state of the referred object can change)

'final' local variables


What are 'final' local variables?

```
class Bicycle {  
public void replaceParts(Part[] parts) {  
final Part[] currentParts = getParts();
```

```
currentParts[0] = new Part();
```

```
currentParts = new Part[3];  
}  
}
```

OK - The state of the object is changed not its identity

 The final local variable currentParts cannot be assigned. It must be blank and not using a compound assignment

Compilation error

'final' attributes

What are 'final' attributes?


```
class Bicycle {  
    private final Gear gear;  
  
    public Bicycle() {  
        this.gear = new Gear();  
    }  
}
```

final attributes can be assigned only once (the state of the referred object can change)

'final' attributes

What are 'final' attributes?


```
class Bicycle {  
    private final Gear gear;  
  
    public Bicycle() {  
        this.gear = new Gear();  
    }  
  
    public void replaceParts(final Part[] parts) {  
        this.gear.changeColor();    OK - The state of the object is  
                                     changed not its identity  
    }  
  
    this.gear = null;  
}  
}
```

 The final field Bicycle.gear cannot be assigned
Compilation error

'final' attributes

What are 'final' attributes?

```
class Bicycle {  
    private final Gear gear;  
  
    public Bicycle() {  
        // this.gear = new Gear();  
    }  
}
```

 The blank final field gear may not have been initialized

Compilation error

final attributes must be initialized

Strings concatenation

Any problem with the following code?

```
public String toString(int[] numbers) {
    String result = "";
    for (int i = 0; i < numbers.length-1; i++) {
        result += numbers[i];
        result += ", ";
    }
    result += numbers[numbers.length-1];
    return result;
}
```


Strings concatenation

Any problem with the following code?

```
public String toString(int[] numbers) {  
    String result = "";  
    for (int i = 0; i < numbers.length-1; i++) {  
        result += numbers[i];  
        result += ", ";  
    }  
    result += numbers[numbers.length-1];  
    return result;  
}
```

How many String objects are created?

*Avoid using += for concatenating
Strings within loops*

Strings concatenation

Use a StringBuilder/StringBuffer instead

```
public String toString(int[] numbers) {
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < numbers.length-1; i++) {
        result.append(numbers[i]);
        result.append(", ");
    }
    result.append(numbers[numbers.length-1]);
    return result.toString();
}
```

Floating point operations

What does this print?

```
System.out.println(new BigDecimal(0.1).toPlainString());
```

Floating point operations

What does this print?

```
System.out.println(new BigDecimal(0.1).toPlainString());
```

```
0.100000000000000000000055511151231257827021181583404541015625
```

Don't assume that floating point numbers are exact representations of mathematical values!

Floating point operations

What does this print?

```
double f = 2.00;  
double g = 1.10;
```

```
System.out.println(f - g == 0.90);
```

false

```
System.out.println(2.00 - 1.10 == 0.8999999999999999);
```

true

0.899999999999999911182158029987476766109466552734375

Floating point operations may lose precision

Floating point operations

What does this print?

```
double r = 0.1 + 0.2;
```

```
System.out.println(r == 0.3);
```

```
System.out.println(new Double(r).equals(new Double(0.3)));
```

false

```
double epsilon = 0.00000001;
```

```
System.out.println(Math.abs(r-0.3) < epsilon);
```

true

Never compare double/float values using ==

Floating point operations

What does this print?

```
for(double balance = 10; balance != 0; balance -=0.1){  
    System.out.println(balance);  
}
```

infinite loop

The balance will never be exactly 0

Floating point operations

What does this print?

```
for(double balance = 10; balance != 0; balance -=0.1){  
    System.out.println(balance);  
}
```

```
for(double balance = 10; balance > 0; balance -=0.1){  
    System.out.println(balance);  
}
```


Try-catch

Does try or finally return?

```
class A {  
    int m() {  
        try { return 1; }  
        catch (Exception err) { return 2; }  
        finally { return 3; }  
    }  
}
```

Prints 1, 2, or 3?

```
A a = new A();  
System.out.println(a.m());
```

3

The finally clause always takes precedence!

Don't use 'return' in the finally clause.

Try-catch - Working with files

Anything wrong with this code?

```
try {  
    BufferedReader reader = new BufferedReader(new FileReader(filePath));  
    print(reader.readLine());  
    reader.close();  
} catch (IOException ex) {  
    ex.printStackTrace();  
}
```

If there is an IOException the stream will not be closed

Try-catch - Working with files

Always close the file handler

```
BufferedReader reader = null;
try {
    reader = new BufferedReader(new FileReader(filePath));
    print(reader.readLine());
} catch (IOException ex) {
    ex.printStackTrace();
} finally {
    if (reader != null) {
        try {
            reader.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Try-catch - Working with files

Use try-with-resources to automatically close resources

```
try(BufferedReader reader = new BufferedReader(new FileReader(filePath))){
    reader.readLine();
} catch (IOException e) {
    e.printStackTrace();
}
```

Thank you!