



While Loops, Wrappers, and Graphical I.O.

Feb 28, 2012

CMPSCI 121, Spring 2008

Introduction to Problem Solving with Computers

Prof. Learned-Miller

Logistics

- Chapter 6 exercises.
 - Due Mar 1 Thursday
- Dice Problem.
 - Due Mar 2 Friday.
- Midterm, next Wednesday (March 7th)
 - 7:15 PM to 8:45 PM

Pasting into OWL

Finish the `doubleEachDay` method below.

```
public int doubleEachDay(double jackpot) {  
    double amount = 0.01;  
    int numDays = 0;
```



Check Answer

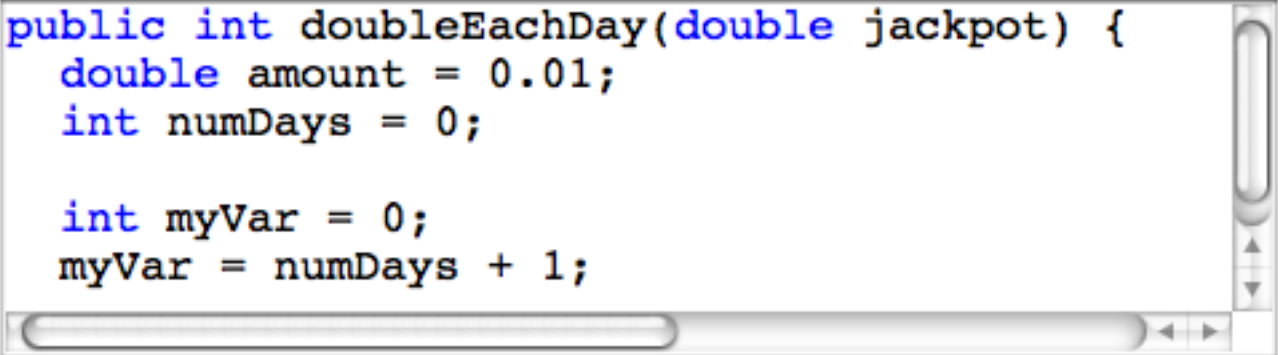
1:1

```
    return numDays;  
} //end method
```

Pasting into OWL

Finish the `doubleEachDay` method below.

```
public int doubleEachDay(double jackpot) {  
    double amount = 0.01;  
    int numDays = 0;
```



```
public int doubleEachDay(double jackpot) {  
    double amount = 0.01;  
    int numDays = 0;  
  
    int myVar = 0;  
    myVar = numDays + 1;
```

Check Answer

7:6

```
    return numDays;  
} //end method
```

Other OWL issues

- Problems: OWL should work well
 - Let us know if your code works in Dr. Java but not in OWL.
- Programming assignments:
 - Make sure it runs in Dr. Java.
 - As programs get more complicated, OWL has a harder time with them.

Common question

- Can I put 2 class definitions in one file?
 - NO, always one file per class.

Today

- While loops
 - Almost like for loops, but slightly different
- Wrapper classes
 - Mostly used for converting numbers to strings, and strings to numbers
- Graphical IO
 - like “scanner” class, but with pop up windows.
- Switch statements:
 - like “If” statements, but slightly fancier.

While loops

```
int j = 1;
while (j <= 5){
    System.out.println(j);
    j = j + 1;
}
```


While loops

```
int j = 1;  
while (j <= 5){  
    System.out.println(j);  
    j = j + 1;  
}
```

While loops

```
int j = 1;
while (j <= 5){
    System.out.println(j);
    j = j + 1;
}
```

While loops

```
int j = 1;  
while (j <= 5){  
    System.out.println(j);  
    j = j + 1;  
}
```

While loops vs. For loops

- Either can be used!
- It's a **style** issue:
 - Sometimes it's more natural to use one or the other.
- for loops tend to be used when
 - We know ahead of time when we will end
 - “from a to z”
 - “from 1 to 10 by twos”
- While loops tend to be used when
 - Termination condition is more complicated

While vs. For

```
for (int i=0; i< 10; i++) {  
    System.out.println("Random number i: "+Math.rand());  
}
```

```
double foo=Math.rand();  
while (foo<.5) {  
    System.out.println("Got another number less than 0.5");  
    foo=Math.rand();  
}
```

While loops: moving the increment

```
int j = 0;
while (j < 5){
    j = j + 1;
    System.out.println(j);
}
```

Counting downwards

```
int j = 9;
while (j >= 5){
    System.out.println(j);
    j = j - 1;
}
```

Write the equivalent “for” loop

```
int j = 9;
while (j >= 5){
    System.out.println(j);
    j = j - 1;
}
```



```
int j;  
for (j=9; j>=5; j--) {  
    System.out.println(j);  
}
```

Private attributes

```
public class Palindrome{  
    private String letters;  
  
    public Palindrome(String s){  
        letters = s;  
    }  
  
    public boolean palCheck(){  
        int left = 0; // position of first char  
        int right = letters.length() - 1; // position of last char  
        boolean ok = true; // assume a palindrome  
        while(left < right){  
            if (letters.charAt(left) != letters.charAt(right))  
                ok = false;  
            left++;  
            right--;  
        }  
        return ok;  
    }  
}
```

Can't access private attribute directly

```
>  
> Palindrome p=new Palindrome("foo");  
> p.letters
```

IllegalAccessException: Class koala.dynamicjava.interpreter.EvaluationVisitor can not access a member of class Palindrome with modifiers "private"

```
at sun.reflect.Reflection.ensureMemberAccess(Reflection.java:65)  
at java.lang.reflect.Field.doSecurityCheck(Field.java:954)  
at java.lang.reflect.Field.getFieldAccessor(Field.java:895)  
at java.lang.reflect.Field.get(Field.java:357)
```

```
>
```

Where's the "private"?!?!

```
public class Palindrome{  
    String letters;  
  
    public Palindrome(String s){  
        letters = s;  
    }  
  
    public boolean palCheck(){  
        int left = 0; // position of first char  
        int right = letters.length() - 1; // position of last char  
        boolean ok = true; // assume a palindrome  
        while(left < right){  
            if (letters.charAt(left) != letters.charAt(right))  
                ok = false;  
            left++;  
            right--;  
        }  
        return ok;  
    }  
}
```

Declare an Instance of the Class

```
>  
> Palindrome p = new Palindrome("foo");  
...
```

Accessing Attributes from Outside the Class!!!

```
>  
> Palindrome p = new Palindrome("foo");  
> p.letters  
"foo"  
>
```

Can change public attributes directly (no set method required)

```
>  
> Palindrome p = new Palindrome("foo");  
> p.letters  
"foo"  
>  
> p.letters="I've changed the attribute!!!"  
"I've changed the attribute!!!"  
>  
> p.letters  
"I've changed the attribute!!!"  
>
```

Usually frowned upon in java programming
Considered safer to use private attributes with
set methods

Wrapper classes

- When you want to treat a fundamental type (such as int, double, boolean) like a class object.
- Integer
- Double
- Boolean

Wrapper classes (continued)

- Main uses for wrapper classes come later
- static method can be used for conversion
- How do I get 123 out of “123” ?

```
Integer.parseInt(“123”);
```

“Fancy” Input/Output: JOptionPane

```
1  import javax.swing.JOptionPane;
2
3  public class SickorWell{
4
5      public static void main(String[] args){
6          String myTemp;
7          double temp;
8          myTemp = JOptionPane.showInputDialog("Enter your temperature");
9          temp = Double.parseDouble(myTemp);
10         if(temp > 98.6)
11             JOptionPane.showMessageDialog(null, "you are sick!");
12         else
13             JOptionPane.showMessageDialog(null, "you are well!");
14     }
15 }
```

JOptionPane

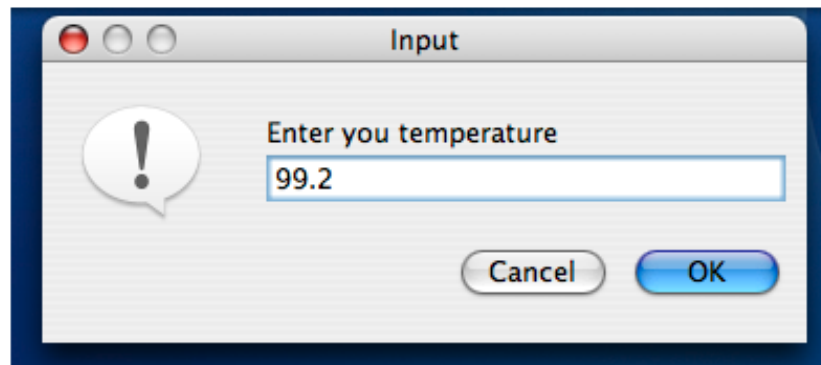
```
1  import javax.swing.JOptionPane;
2
3  public class SickorWell{
4
5      public static void main(String[] args){
6          String myTemp;
7          double temp;
8          myTemp = JOptionPane.showInputDialog("Enter your temperature");
9          temp = Double.parseDouble(myTemp);
10         if(temp > 98.6)
11             JOptionPane.showMessageDialog(null, "you are sick!");
12         else
13             JOptionPane.showMessageDialog(null, "you are well!");
14     }
15 }
```

JOptionPane: Static methods

```
1  import javax.swing.JOptionPane;
2
3  public class SickorWell{
4
5      public static void main(String[] args){
6          String myTemp;
7          double temp;
8          myTemp = JOptionPane.showInputDialog("Enter your temperature");
9          temp = Double.parseDouble(myTemp);
10         if(temp > 98.6)
11             JOptionPane.showMessageDialog(null, "you are sick!");
12         else
13             JOptionPane.showMessageDialog(null, "you are well!");
14     }
15 }
```

JOptionPane: Static methods

```
1  import javax.swing.JOptionPane;
2
3  public class SickorWell{
4
5      public static void main(String[] args){
6          String myTemp;
7          double temp;
8          myTemp = JOptionPane.showInputDialog("Enter you temperature");
9          temp = Double.parseDouble(myTemp);
10         if(temp > 98.6)
11             JOptionPane.showMessageDialog(null, "you are sick!");
12         else
13             JOptionPane.showMessageDialog(null, "you are well!");
14     }
15 }
```



JOptionPane: Static methods

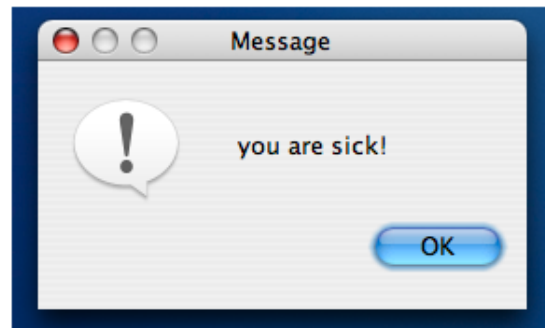
```
1  import javax.swing.JOptionPane;
2
3  public class SickorWell{
4
5      public static void main(String[] args){
6          String myTemp;
7          double temp;
8          myTemp = JOptionPane.showInputDialog("Enter your temperature");
9          temp = Double.parseDouble(myTemp);
10         if(temp > 98.6)
11             JOptionPane.showMessageDialog(null, "you are sick!");
12         else
13             JOptionPane.showMessageDialog(null, "you are well!");
14     }
15 }
```

JOptionPane: Static methods

```
1  import javax.swing.JOptionPane;
2
3  public class SickorWell{
4
5      public static void main(String[] args){
6          String myTemp;
7          double temp;
8          myTemp = JOptionPane.showInputDialog("Enter your temperature");
9          temp = Double.parseDouble(myTemp);
10         if(temp > 98.6)
11             JOptionPane.showMessageDialog(null, "you are sick!");
12         else
13             JOptionPane.showMessageDialog(null, "you are well!");
14     }
15 }
```

JOptionPane: Static methods

```
1  import javax.swing.JOptionPane;
2
3  public class SickorWell{
4
5      public static void main(String[] args){
6          String myTemp;
7          double temp;
8          myTemp = JOptionPane.showInputDialog("Enter your temperature");
9          temp = Double.parseDouble(myTemp);
10         if(temp > 98.6)
11             JOptionPane.showMessageDialog(null, "you are sick!");
12         else
13             JOptionPane.showMessageDialog(null, "you are well!");
14     }
15 }
```



Switch

```
switch( variable_name ){  
case case_1:  
    statement_1; ←  
    break;  
case case_2:  
    statement_2; ←  
    break;  
default:  
    statement_d; ←  
}
```

Jump to here if (variable_name == case_1)

Jump to here if (variable_name == case_2)

Jump to here for all other cases

NOTE: Use **break** (optional) if you want to jump out of switch after *case_1* and before *case_2* statements

break

- Break statement takes you out of the current loop you are in (for/while)
- Useful when you don't want to continue to run the loop if a condition is met
 - For instance, when checking palindrome, if you see a character pair that is mismatched, you don't need to continue checking subsequent pairs
 - Think of how you could have used this in the “prefix” problem

```
class SwitchExample
{
    public void rateLetter(char ch)
    {
        switch(ch)
        {
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u': System.out.println("vowel");
                break;
            case 'y': System.out.println("vowel or consonant");
                break;
            default: System.out.println("consonant");
        }
    }
}
```