Computer Science II

Professor Tom Ellman

Lecture 5
Iteration

• Executing a block of code over and over.

• Implemented using *loops* in Java.
  – Zero or more iterations: *while* loop.
  – One or more iterations: *do-while* loop.
  – Indexed iteration: *for* loop.
General Form of `while` Loop

```plaintext
while (<Condition>)
{
    <Statement1>
    <Statement2>
    . . .
    <StatementN>
}
```
Semantics of **while** Loop

1. Evaluate <Condition>.
2. If <Condition> is *false*, then go to step 5.
3. Otherwise (<Condition> is *true*) execute <Statement1> … <StatementN>.
4. Go to step 1.
5. Execute statements following the **while** loop.
Semantics of \texttt{while} Loop

\begin{itemize}
  \item \texttt{Condition}
  \item \texttt{Entry}
  \item \texttt{Statement1}
  \item \texttt{...}
  \item \texttt{StatementN}
  \item \texttt{Exit}
  \item \texttt{true}
  \item \texttt{false}
\end{itemize}
Square Root Program Specification

• Given a double, \textit{number}.
• Given a double error threshold, \textit{epsilon}.
• Find the square root of \textit{number}.
• To within a tolerance of \textit{epsilon}.
Square Root Program Pseudo-Code

1. Initialize bounds and guess.

2. While ( Bounds are Too Loose )
   a. Determine if guess is too high or too low.
   b. Revise bounds and guess accordingly.

3. Print out guess.
package squarerootiterative;

public class SquareRootIterative {
    public static double number = 2.0;
    public static double epsilon = 1E-15;
    public static void main(String[] args) {
        double lower = 0.0;
        double upper = number;
        double guess = number/2.0;
        while ( (upper-lower)/2.0 > epsilon ) {
            double square = guess*guess;
            if (square > number)
                upper = guess;
            else lower = guess;
            guess = (lower + upper)/2.0;
        }
        System.out.println("The square root of " + number + " is: " + guess);
    }
}

Note: There’s a bug here. Can you find it?
General Form of **do**...**while** Loop

do

{<br>
  <Statement1>
  <Statement2>
  ...<br>
  <StatementN><br>
}<br>
while (<Condition>)
Semantics of \texttt{do...while} Loop

1. Execute \texttt{<Statement1>} \ldots \texttt{<StatementN>}.  
2. Evaluate \texttt{<Condition>}.  
3. If \texttt{<Condition>} is \texttt{false}, then go to step 5.  
4. Otherwise (\texttt{<Condition>} is \texttt{true}) go to step 1.  
5. Execute statements following the \texttt{do...while} loop.
Semantics of **do...while** Loop

```plaintext
Entry

<Statement1>

... 

<StatementN>

<Condition>

true

false

Exit
```
Guessing Game Program

- Program randomly selects a number in the range [1…100].
- User repeatedly tries to guess the number.
- After each guess, the program reports “Too Low”, “Too High” or “You got it!”.
- Program terminates when user guesses the number.
Guessing Game Program Pseudo-Code

1. Do some initialization.

2. Display message describing the game.

3. Pick a random number.

4. do {
    A. Read the user’s guess.
    B. Display a high, low or correct message.
}

while (The User’s Guess is Wrong)

4. Do some finalization.
package guessinggame;
import acm.program.*;
import java.util.Random;

public class GuessingGame extends ConsoleProgram {

    // ... Omitted ...

    public static void main(String[] args) {
        new GuessingGame().start();
    }

}
public void run() {
    println("I am thinking of a number from 1 to 100.");
    println("Can you guess it?");
    Random random = new Random();
    int secretNumber = 1 + Math.abs(random.nextInt()) % 100;
    int guess;
    do {
        guess = readInt();
        if (guess < secretNumber)
            print("Too low. Guess again: ");
        else if (guess > secretNumber)
            print("Too high. Guess again: ");
        else println("You got it!");
    } while (guess != secretNumber);
}
General Form of \texttt{for} Loop

for (<Initialization>; <Condition>; <Increment>)
{
    <Statement1>
    <Statement2>
    ... 
    <StatementN>
}
Semantics of \texttt{for} Loop

1. Execute \texttt{<Initialization>}.  
2. Evaluate \texttt{<Condition>}.  
3. If \texttt{<Condition>} is \texttt{false}, then go to step 7.  
4. Otherwise (\texttt{<Condition>} is \texttt{true}) execute \texttt{<Statement1> \ldots <StatementN>}.  
5. Execute \texttt{<Increment>}.  
6. Go to step 2.  
7. Execute statements following the \texttt{for} loop.
Semantics of \textbf{for} Loop
Draw Tower Program

• Draw a tower, composed of a number of square cells.

• Use a for loop in which an index variable keeps track of the floor on which a cell should be drawn.
package drawtower;
import acm.program.*;
import acm.graphics.*;
public class DrawTower extends GraphicsProgram {
    protected static int NUM_FLOORS = 7;
    protected static int SIZE = 50;
    protected static int SPACE = 100;
    public void run() {
        for (int i=0; i<NUM_FLOORS; i++) {
            GRect rect = new GRoundRect(SPACE,SPACE+SIZE*i,SIZE,SIZE);
            rect.setFillColor(Color.GREEN);
            rect.setFilled(true);
            add(rect);
        }
    }
    public static void main(String[] args) {
        new DrawTower().start();
    }
}
Draw Tower Program
Nested for Loops

for (<Initialization1>; <Condition1>; <Increment1>)
{
    for (<Initialization2>; <Condition2>; <Increment2>)
    {
        <Statement1>
        <Statement2>
        . . .
        <StatementN>
    }
}

}
**DrawCheckerBoard Program**

- Draw a picture of an 8 by 8 checkerboard.

- Use two nested *for* loops.
  - Arrange for the outer loop to iterate over rows.
  - Arrange for the inner loop to iterate over columns.

- Use the modulus `%` operator to determine whether to color a given square red or black.
package drawcheckerboard;

import acm.program.*;
import acm.graphics.*;
import java.awt.*;

public class DrawCheckerBoard extends GraphicsProgram {

    protected static final int BOARD_SIZE = 8;
    protected static final int SQUARE_SIZE = 50;

    // ... Omitted ...

    public static void main(String[] args) {
        new DrawCheckerBoard().start();
    }

}
public void run() {
    for (int row = 0; row < BOARD_SIZE; row++) {
        for (int col = 0; col < BOARD_SIZE; col++) {
            GRect rect = new GRect((1+col)*SQUARE_SIZE,
                                    (1+row)*SQUARE_SIZE,
                                    SQUARE_SIZE,
                                    SQUARE_SIZE);
            rect.setColor( (row+col)%2==0 ? Color.RED : Color.BLACK);
            rect.setFilled(true);
            add(rect);
        }
    }
}
DrawCheckerBoard Program
package squarerootrecursive;

public class SquareRootRecursive {
    public static double number = 2.0;
    public static double epsilon = 1E-15;

    public static void main(String[] args) {
        double lower = 0.0;
        double upper = number;
        double guess = number/2.0;
        double root = squareRoot(guess,lower,upper);
        System.out.println("The square root of "+number+" is: "+root);
    }
    // ... Omitted ...
}
private static double squareRoot(double guess, double lower, double upper) {
    if ((upper-lower)/2.0 <= epsilon) {
        return guess;
    } else {
        double square = guess*guess;
        if (square > number) {
            return squareRoot((lower + guess)/2.0,lower,guess);
        } else {
            return squareRoot((guess + upper)/2.0,guess,upper);
        }
    }
}
package drawpyramid;
import acm.program.*;

public class DrawPyramid extends GraphicsProgram {

    private static final int NUM_STEPS = 5;
    private static final double X_POS = 100.0;
    private static final double Y_POS = 600.0;
    private static final double X_STEP = 100.0;
    private static final double Y_STEP = 100.0;

    public void run() {
        Pyramid myPyramid = new Pyramid(NUM_STEPS,X_POS,Y_POS, X_STEP, Y_STEP);
        add(myPyramid);
    }

    public static void main(String[] args) {
        new DrawPyramid().start();
    }
}
package drawpyramid;
import acm.graphics.*;
import java.awt.Color;

public class Pyramid extends GCompound {

    private GRoundRect base;
    private Pyramid subPyramid;

    public Pyramid(int n, double x, double y, double xStep, double yStep) {
        base = new GRoundRect(x,y-yStep,xStep*(2*n-1),yStep);
        base.setFillColor(Color.ORANGE);
        base.setFilled(true);
        add(base);
        if (n==1) subPyramid = null;
        else {
            subPyramid = new Pyramid(n-1, x+xStep, y-yStep, xStep, yStep);
            add(subPyramid);
        }
    }
}

Translate \texttt{do...while} into \texttt{while}:

\begin{verbatim}
do
  {
    <Statements>
  }
while (<Condition>)
\end{verbatim}
Translate **while** into **do-while**:

```
while (<Condition>)
{
    <Statements>
}
```

```
if (<condition>)
{
    do
    {
        <Statements>
    }
while (<Condition>)
```
Translate **for** into **while**:

```plaintext
for (<Init>; <Cond>; <Inc>)
{
  <Statements>
}
```

```plaintext
<Init>
while (<Cond>)
{
  <Statements>
  <Inc>
}
```
Translate \texttt{while} into \texttt{for}:

\begin{align*}
\text{while (}<\text{Condition}>\text{)} & \rightarrow \text{for ( ; }<\text{Condition}>\text{ ; )} \\
\{ & \\
\text{<Statements>} & \\
\} & \\
\{ & \\
\text{<Statements>} & \\
\} & 
\end{align*}
Equivalence of Loops

- Each type of loop can be translated into either of the other two types of loop.
- Suppose Java were changed to allow only `while` loops, or only `do...while` loops, or only `for` loops.
- The new Java language would be capable of exactly the same computations as the original Java language.
- Many programs written in the new Java language would be more confusing and difficult to read.