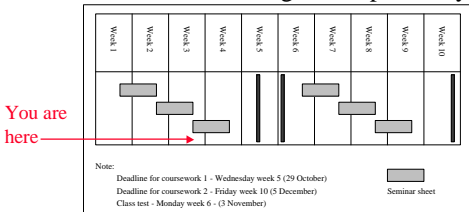


## Lecture 5

- Coursework: documenting your solutions
  - Keep a record of your solutions (Ex7.java etc.)
  - Make sure you comment your code
  - Keep a record of any non-program answers (e.g. Ex. 8)
- Seminars: week 3 begins at 5pm today



## Robot controller skeleton

```
import uk.ac.warwick.dcs.maze.logic.IRobot;

public class DumboController
{
    public void controlRobot(IRobot robot) {

        int randno;
        int direction;
        // Select a random number (0..3 inclusive)
        randno = (int) Math.round(Math.random()*3);
        // Convert this to a direction
        if (randno == 0) direction = IRobot.LEFT;
        else if (randno == 1) direction = IRobot.RIGHT;
        else if (randno == 2) direction = IRobot.BEHIND;
        else direction = IRobot.AHEAD;

        robot.face(direction); // face
        robot.advance();      // move
    }
}
```

## Last lecture

- Control flow
  - Introduced choice using 3 new Java expressions

<pre>{   s1   if (x)   { s2 }   s3 }</pre> <p>s1, s2, s3 if x s1, s3 if !x</p>	<pre>{   s1   if (x)   { s2 }   else   { s3 }   s4 }</pre> <p>s1, s2, s4 if x s1, s3, s4 if !x</p>	<pre>{   s1   switch (x)   {     case a : s2; break;     case b : s3; break;     ...     default : s4   }   s5 }</pre>
--	--	--

## Control statements: Repetition

- Modelling fixed number of repetitive actions
  - Print “stephen is cool” 25 times
  - Chant “Tyson is a wuss” three times
  - Run the payroll function a million times
- The number of repetitions may be unknown
  - Walk North until the tree is found
  - Stir the sauce until it thickens
  - Choose a random direction until a WALL-free route is found
- First is known as *bounded* repetition; the second as *unbounded* repetition

## Do we need bounded repetition?

- Consider the problem of printing 3 asterisks
 

```
System.out.print("***");
System.out.print("***");
System.out.print("***");
```
- Consider the problem of printing 4 asterisks
 

```
System.out.print("***");
System.out.print("***");
System.out.print("***");
System.out.print("***");
```
- Consider the problem of printing  $N$  asterisks
 

```
System.out.print("***"); ... ?
```

## Bounded repetition

Need some way of modelling this repetitive action in our code

- that can be based on  $N$
- where we might have a counter, beginning at 0
- and we increase the counter by 1
- and where we repeat some code each time, i.e.
 

```
System.out.print("***");
```
- until our counter reaches the value of  $N$
- then we stop

## The Java for statement

- Syntax

```
for (statement1; booleanExpression; statement2)
{
    // Loop body
}
```

- **statement1:** initialisation of counter (control) variable
  - **statement2:** adjustment of counter (control) variable
  - **booleanExpression:** loop guard (ie. under what conditions should the repetition continue)
- Curley brackets are optional if the loop body only contains one program statement.

## The Java for statement

### Semantics

1. The control variable is initialized
2. If the loop guard evaluates to true
  3. The statements inside the loop body are executed
  4. The control variable is adjusted
  5. Go back to 2
6. else the loop guard evaluates to false and the loop ends

## Bounded repetition: for loop

- Java 'for loop' example

```
int n; // n is our counter (control) variable
for ( n = 0; n != N; n = n + 1 )
{
    System.out.print("*");
}
```

- When  $N=4$

value of $n$ :	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$
loop guard:	true	true	true	true	false
output:	*	*	*	*	

## The for statement: Examples

- Print "stephen is cool" 25 times

```
for ( n = 0; n != 25; n = n + 1 ) // An up loop
{
    System.out.println("stephen is cool");
}
```

- Chant "Tyson is a wuss" three times

```
for ( n = 3; n != 0; n = n - 1 ) // A down loop
{
    Chant(Tyson is a wuss);
}
```

- Run the payroll function a million times

```
for ( n = 0; n != 2000000; n = n + 2 ) // Up-loop step 2
{
    Payroll();
}
```

## Unbounded repetition

- While input is not -1 add input and then display running total. Eg.

```
Input number (-1 to finish) 5
Total = 5
Input number (-1 to finish) 6
Total = 11
Input number (-1 to finish) 22
Total = 33
Input number (-1 to finish) -1
```

The difference here is that we can not be sure on the number of repetitions

## Unbounded repetition

- Information the programmer needs to know

- Where does the repetition start  
`input = IO.readInt("Input number (-1 to finish) ");`
- What happens at the end of one iteration  
`input = IO.readInt("Input number (-1 to finish) ");`
- When does the repetition stop  
`input == -1`
- What statement(s) is repeated  
`total = total + input;`  
`System.out.println("Total = " + total);`

## The Java while statement

- Syntax

```
// control variable is initialized
while ( booleanExpression )
{
    // Loop body
    // control variable is adjusted
}
```

– *booleanExpression*: loop guard (ie. under what conditions should the repetition continue)

- Brackets are optional if the loop body only contains one program statement.

## The Java while statement

### Semantics

1. The control variable is initialized
2. If the loop guard evaluates to true
  3. The statements inside the while loop are executed
  4. The control variable is adjusted
  5. Go back to 2
6. else the loop guard evaluates to false and the while loop ends

## Unbounded repetition: while loop

Java ‘while loop’

```
int total = 0;           // running total variable
int input;              // control variable
input = IO.readInt("Input number (-1 to finish) ");
while ( input != -1 )
{
    total = total + input;
    System.out.println("Total = " + total);
    input = IO.readInt("Input number (-1 to finish) ");
}
```

## The while statement: Examples

- Walk North until the tree is found

```
boolean found_it = treeFound();
while ( !found_it )
{
    walkNorth();
    found_it = treeFound(); // Update the control variable
}
```

- Stir the sauce until it thickens

```
while ( !sauceThick() )
{
    stirSauce();
}
```

## Unbounded repetition: do - while loop

- Sometimes you want to execute the loop body before you check the loop guard

```
do {
    pick a card
} while ( the card is not the ace of hearts );
```

## The do - while statement

- Syntax

```
// control variable is initialized
do {
    // Loop body in which control variable
    // is adjusted
} while ( booleanExpression );
```

– *booleanExpression*: loop guard

- Brackets are optional if the loop body only contains one program statement.

## The do - while statement

- Semantics
  1. The statements inside the do-while loop are executed
  2. If the loop guard evaluates to true
    3. Go back to 1
  4. else the loop guard evaluates to false and the do-while loop ends
- Difference between a while and do-while
  - do-while ensures that the loop body is executed at least once

## The do-while statement: Example

- Choose a random direction until a WALL-free route is found

```
do
{
    // Some code that chooses a random direction
} while ( robot.look(direction) == IRobot.WALL );
```

- Is this the right statement to use?
  - Yes, because you have to choose a direction before you can decide whether or not it is any good.

## break and continue

- break - immediately ends the innermost loop which contains it

```
for ( i=1; i<=10; i = i + 1 )
{
    if (i==5) break;
    System.out.println(i);
}
```

- continue - ends the current iteration; transfers control to next iteration of loop

```
for ( i=1; i<=10; i = i + 1 )
{
    if (i==5) continue;
    System.out.println(i);
}
```

## A question of correctness

- Back to the for-loop example

```
int n; // n is our counter (control) variable
for ( n = 0; n != N; n = n + 1 )
{
    System.out.print("***");
}
```

- Does this code always work?

## A question of correctness

- A program to print half as many asterisks

```
n = 0; // using some counter n
while ( n != N )
{
    System.out.print("***");
    n = n + 2;
}
```

## Loop invariants

- Something that must be true initially, and then true after each iteration of the loop
- Previous slide
  - It must be the case that N is even
  - The loop body must maintain this condition
- For-loop example
  - It must be the case that N >= 0
  - The loop body must maintain this condition
- GCD example
  - It must be the case that a > 0 and b > 0
  - The loop body must maintain this condition
    - If a>0 and b>0 and a==b, then end loop and print one of them
    - If a>0 and b>0 and a>b, then a = a-b > 0, repeat loop
    - If a>0 and b>0 and b>a, then b = b-a > 0, repeat loop