

Same statements, different sequence

```

robot.forward();
robot.forward();
robot.right();
robot.forward();
robot.right();
robot.right();
robot.forward();
    
```

Start

End up in a different place.

7 Copyright © 2003, Graham Roberts Department of Computer Science

Same program, different starting state

```

robot.forward();
robot.forward();
robot.right();
robot.forward();
robot.right();
robot.right();
robot.forward();
    
```

Start

Facing a different direction at start.

8 Copyright © 2003, Graham Roberts Department of Computer Science

Going to the door

```

robot.forward();
robot.forward();
robot.forward();
robot.right();
robot.forward();
robot.left();
robot.forward();
    
```

Start

Easy - Problem solved??

9 Copyright © 2003, Graham Roberts Department of Computer Science

What if ?

- ... the robot starts off facing in a different direction?
- ... the robot is put in a different starting position or a different room?
- ... the door moves?

10 Copyright © 2003, Graham Roberts Department of Computer Science

The program context

- The program only solves the problem (getting to the door) in very specific circumstances.
- Change the circumstances, rewrite the program...

Start

11 Copyright © 2003, Graham Roberts Department of Computer Science

Too much hard work!

- In fact, *you* have to do all the work of solving the problem.
- The robot just does exactly what you command it to.
- No way! Let's get the robot to do some of the hard work.

12 Copyright © 2003, Graham Roberts Department of Computer Science

What we would really like

- A program that can guide the robot to the door in *any* room, starting in *any* position.
- A program that *doesn't* have to be changed for every specific context.

Help!! That's hard to do...

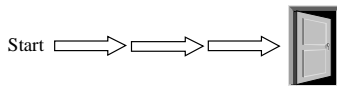
Too hard – simplify

- Let's try to solve a simpler problem first.
- If we can solve that, hopefully we can learn more about solving the harder problem.

Strategy: Look for a sub-problem and solve it first.

First go

- Assume the robot is always facing in the direction of the door.
- Assume there are no obstacles between the robot and the door.



Easy?

- Program is just: robot.forward(); robot.forward(); robot.forward()...
- But how many forwards?
- Depends on how far away door is.
- Snag: we still have to change the program for every start position.

Solution

- We need to be able to say: “Keep moving forward while you have not reached the door”.
- But we need additional commands to do this.

A While Loop

```
while (!robot.atDoor())  
{  
  robot.forward();  
}
```

Loop Body

- Called a “loop” as it loops around!
- Keep looping while a condition is *true*.

Details

- *while* is a *sequencing* command providing *iteration* or *repetition*.
- *atDoor* performs a test that has a *boolean* value.
- *!* meaning *not*, is a *logical operator* that is applied to a boolean value.

19 Copyright © 2003, Graham Roberts

Department of Computer Science



A Result!

- We now have a program that will move the robot to the door, regardless of the starting position.
- Providing it is facing the door.
- Providing there are no obstacles.

20 Copyright © 2003, Graham Roberts

Department of Computer Science



A Harder Problem

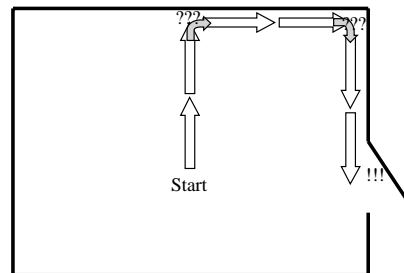
- We've solved our simple problem, so let's make it a bit harder and see what happens.
- Assume robot can start facing any direction.
- Assume robot is in a room with four walls, one door and no obstacles.

21 Copyright © 2003, Graham Roberts

Department of Computer Science



Brainstorming!!



22 Copyright © 2003, Graham Roberts

Department of Computer Science



An idea!!

1. Move forward until robot finds the wall.
2. Turn right and follow wall.
3. Stop if door found.
4. If another wall is found, repeat 2,3,4.

Now turn this into commands!

23 Copyright © 2003, Graham Roberts

Department of Computer Science



Try another program

```
while (robot.canMoveForward())
{
    robot.forward();
}
robot.right();
while (robot.canMoveForward())
{
    ... Hmmm, what happens here?
}
```

24 Copyright © 2003, Graham Roberts

Department of Computer Science



Rethink...

```

while (!robot.atDoor())
{
    while (robot.canMoveForward())
    {
        robot.forward();
    }
    robot.right();
}
    
```

} A *nested* while loop.

This works OK – doesn't it?

25 Copyright © 2003, Graham Roberts Department of Computer Science

Find an example that *fails*

Start

26 Copyright © 2003, Graham Roberts Department of Computer Science

Examine evidence...

- The *algorithm* fails unless door is at corner.
- But, robot does go past the door.
- Every time the robot moves forward it should check for the door.

27 Copyright © 2003, Graham Roberts Department of Computer Science

Try again

```

while (!robot.atDoor())
{
    while (!robot.atDoor() && robot.canMoveForward())
    {
        robot.forward();
    }
    robot.right();
}
    
```

&& means AND

A boolean AND expression

But...

28 Copyright © 2003, Graham Roberts Department of Computer Science

How about?

```

while (!robot.atDoor())
{
    robot.right();
    while (!robot.atDoor() && robot.canMoveForward())
    {
        robot.forward();
    }
}
    
```

29 Copyright © 2003, Graham Roberts Department of Computer Science

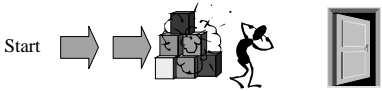
Wow!

- Iteration allows a more general purpose program to be written.
- We have a program that will work in *any* empty room.
- (Providing all our assumptions remain true.)
- What about testing the code – does it *really* work...

30 Copyright © 2003, Graham Roberts Department of Computer Science

Obstacles

- An unexpected pile of bricks...



- Need to divert round the bricks.
- Any need to change the program?
 - Suggestions...

31 Copyright © 2003, Graham Roberts Department of Computer Science

Questions?

32 Copyright © 2003, Graham Roberts Department of Computer Science

Selection

- Suppose the floor is covered in tiles of different colours.
- If robot is on a red tile its red light turns on.
- If robot is on a blue tile its blue light turns on.
- If robot is on white tile no lights are on.

33 Copyright © 2003, Graham Roberts Department of Computer Science

If...

- We want to say:
 - “If the tile is red, then show red”
 - “If the tile is blue, then show blue”
 - “If the tile is white, then show no light”
- A *selection* command is needed.
 - And commands to check tile colour.

34 Copyright © 2003, Graham Roberts Department of Computer Science

Let's Try...

```

while (!robot.atDoor() && robot.canMoveForward())
{
    robot.forward();
    if (robot.onRed())
    {
        robot.turnOnRed();
        robot.turnOffBlue();
    }
    etc.
    ...

```

An if statement

35 Copyright © 2003, Graham Roberts Department of Computer Science

New Commands

- The robot needs:
 - onRed, onBlue and onWhite to check tile colours.
 - turnOnRed, turnOffRed, turnOnBlue, turnOffBlue to control lights.

36 Copyright © 2003, Graham Roberts Department of Computer Science

Results

- We can now write a program to get the robot to find the door of a room.
- And flash its lights as it moves around.
- We've discovered *iteration* and *selection* as being fundamentally necessary to get things working.
- And discovered a set of commands the robot needs to understand.

37

Copyright © 2003, Graham Roberts

Department of Computer Science



Language v. Robot

- While loops and if statements are part of the programming language.
 - They are basic language features.
- Moving, testing for door, controlling lights are part of the robots behaviour.
 - Not part of the programming language.

38

Copyright © 2003, Graham Roberts

Department of Computer Science



Questions?

39

Copyright © 2003, Graham Roberts

Department of Computer Science



Remember – Java and Objects

- Java is object-oriented, so you will encounter many software objects and soon be creating your own.
- You have already seen *out*
 - `System.out.println("hello");`
- And *g*
 - `g.drawLine(...);`

40

Copyright © 2003, Graham Roberts

Department of Computer Science



Objects

- Our robot is really a software *object*.
- It has an external interface.
 - The actions you ask it to perform.
- And an internal implementation.
 - The code that does the work, but which you don't see.

41

Copyright © 2003, Graham Roberts

Department of Computer Science



Objects and Methods

```
robot.forward();
```

- `robot` is the name of the object.
- `forward()` is a *method call*.
 - We use the name method rather than command or action.
- The robot is instructed what to do but actually performs the actions itself.

42

Copyright © 2003, Graham Roberts

Department of Computer Science



Methods

- Each method is a sequence of program statements.
- A robot has a collection of methods, each of which is a different statement sequence.
- Hence, statements are grouped into units rather than being one big long list.

43 Copyright © 2003, Graham Roberts

Department of Computer Science



Responsibilities and Collaborations

- We can also take the point of view that the robot has *responsibilities*:
 - To move, to test, to control lights.
- And *collaborations*:
 - To ask other objects about the floor colour and location of walls and doors.

44 Copyright © 2003, Graham Roberts

Department of Computer Science



Questions?

45 Copyright © 2003, Graham Roberts

Department of Computer Science



Core Language Statements

- The statements (if, while, etc.) we use need to be at the right level of detail.
- Too low level – too awkward to work with.
- Too high level – can't express the amount of detail needed.

46 Copyright © 2003, Graham Roberts

Department of Computer Science



Abstraction

- *abstraction*:
 - A representation or model that includes the important, essential or distinguishing aspects of something while suppressing or ignoring less important, immaterial or diversionary details.
 - Removing distinctions to emphasise commonality.

47 Copyright © 2003, Graham Roberts

Department of Computer Science



Choosing the right abstractions

- Our basic commands, selection and iteration are all abstractions of behaviour.
- They represent the lowest level of abstraction that we generally want to work with.
- (We can go lower – assembly language programming or even direct binary coding...)

48 Copyright © 2003, Graham Roberts

Department of Computer Science



Creating new abstractions

- Programming relies heavily on using and creating abstractions.
- As programs get larger we have to create new abstractions to manage the huge amount of detail involved.
- The robot object is a higher-level abstraction constructed from programming language abstractions.

49 Copyright © 2003, Graham Roberts

Department of Computer Science



Further thoughts...

- We have seen programs with the basic structure of:
 Start -> Do the work -> Stop
 Where we explicitly expect the program to run to a pre-determined conclusion.

Are all programs like that?

50 Copyright © 2003, Graham Roberts

Department of Computer Science



Programs that stop?

- The robot “find the door” program may never stop...
 – but that would be a flaw that requires the program to be fixed.
- What about the drawing programs? How do they stop?
- What about a word processor?
- Or the control system in a car?

51 Copyright © 2003, Graham Roberts

Department of Computer Science



Run for ever...

- In fact, many kinds of programs are designed to run continuously until the *user explicitly stops* them.
- If the program stops any other way something has gone wrong...

52 Copyright © 2003, Graham Roberts

Department of Computer Science



The main loop

- Consider this overall program structure:

```
while (true)
{
    doWork();
    if (userQuits())
    {
        stop();
    }
}
```

53 Copyright © 2003, Graham Roberts

Department of Computer Science



Event driven

- Loop forever, responding to *events*.
- Events are mouse clicks, key presses, timer and so on.

```
while (true)
{
    waitForEvent();
    handleEvent();
    if (userQuits())
    {
        stop();
    }
}
```

54 Copyright © 2003, Graham Roberts

Department of Computer Science



And the drawing programs?

- Your drawing programs are actually event driven, with a main loop.
- You don't see that explicitly – it comes for free as part of the infrastructure of the Java code libraries.

55 Copyright © 2003, Graham Roberts

Department of Computer Science



The Graphical User Interface (GUI)

- All programs that use windows, the mouse and so on, have a GUI.
- These programs are all event driven, with a main loop.
- They are designed to keep running for ever until you tell them to stop.

56 Copyright © 2003, Graham Roberts

Department of Computer Science



Event loop means GUI?

- No. An event driven program doesn't need a GUI.
- Consider control systems (in a car, plane, lift, ATM machine).
- They run continuously, responding to events, until explicitly stopped.
 - Or the power is cut off.

57 Copyright © 2003, Graham Roberts

Department of Computer Science



Summary

- Sequence, iteration, selection.
- Problem solving by decomposing a large problem into simpler smaller problems.
- Starting to think about abstraction.
- The main loop and event driven programming.

58 Copyright © 2003, Graham Roberts

Department of Computer Science

