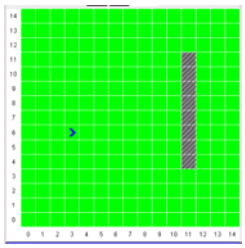
The while statement

Repeating instructions

Problem

Tell a robot to keep moving as long as the way is clear

We need a way to repeat an instruction while a condition is true.
The while statement provides this ability.



The while statement

The while statement is used to repeat an instruction, for example:

```
while(frontIsClear())
    move();
```

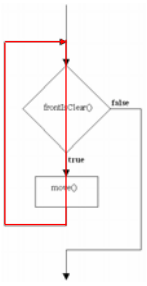
The **`move()`** method will be executed as long as the condition **`frontIsClear()`** is true.

In other words the robot will keep moving forward as long as the way is clear.

The while statement

The **while** statement can be represented by a flowchart

```
while(frontIsClear())
    move();
```



Notice the loop

The while statement

Syntax

```
while(condition)
   statement;
```

Example

```
while(frontIsClear())
   move();
```

Purpose

To execute a statement while a condition is true.

How It Is Executed

- The robot first checks the condition.
- If the condition is true, the robot executes the instruction and then re-executes the entire while loop
- If the condition is false, the robot is finished with the while instruction, and continues by executing instructions that follow the while loop

Runner.java

```
public class Runner extends Robot
{
   void goTilBlocked()
   {
      while(frontIsClear())
         move();
   }
}
```

WhileExample.java

```
public class WhileExample
{
   public static void main(String [] args)
   {
      World wallField = new World();
      Runner jill = new Runner();

      wallField.addWall(11, 4, 8, "north");
      wallField.add(jill, 3, 6, "east");

      jill.goTilBlocked();
   }
}
```

Indentation

- As with the if statement, instructions controlled by a while statement should be indented
- For example

```
while(frontIsClear())
   move();
turnLeft();
```

This is correctly indented

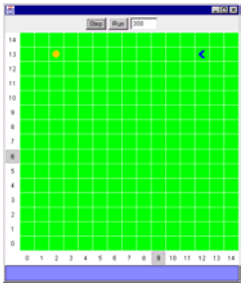
Finding Beepers

- A robot has a beeper somewhere in front of him. Write a while loop to cause the robot to move to the beeper.
- Hint use the beeperPresent() method. And you may also have to use the NOT operator (!).
- You may assume that there is no obstruction between you and the beeper.

Finding Beepers

In this example you want the robot to move forward as long as no beeper is in the cell, i.e. while NOT beeperPresent()

```
while(!beeperPresent())
   move();
```

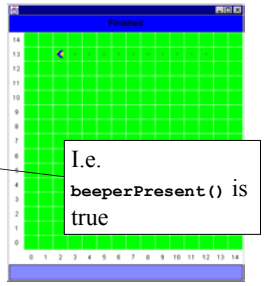


Finding Beepers

```
while(!beeperPresent())
   move();
```

The robot keeps moving until the condition **`!beeperPresent()`** is **false**.

What happens if the beeper is just east of the robot?



Another Example

A cell may contain more than one beeper. Use a **while** loop to pick up all the beepers in a cell

```
while(beeperPresent())
   pickBeeper();
```

What happens when the robot's cell has 2 beepers? zero beepers?

Another while Loop

```
while(beeperPresent())
   turnLeft();
```

What happens when a robot is on a cell with zero beepers? With 2 beepers?

The instruction in the while loop doesn't effect the number of beepers, and therefore the condition beeperPresent() remains true. The loop will never exit. This is called an infinite loop.

Infinite WHILE Loops

- This is called an *infinite loop*; it is a kind of logic error
- The ability to have a robot execute an instruction an unknown number of times has dangers as well as benefits
- There is no danger of an infinite loop with the other instructions
- Check your while instructions carefully to make sure they terminate

Formal Property of WHILE

- When (if) a while statement finishes executing, the condition is known to be false
- If you need to make some condition true, you write something like:

```
while(!condition)
   instruction
```

- For example, to make **`frontIsClear()`** true, write a while loop with the condition **`!frontIsClear()`**

Example

- To ensure that the front is clear a robot would use the condition **`!frontIsClear()`**
- The instruction should be something that can cause the condition to become true. In this case, **`turnLeft()`**

```
while(!frontIsClear())
   turnLeft();
```

How to Make Sure while instructions terminate?

- Each execution of the instruction should bring the robot closer to finishing the loop, otherwise you get an infinite loop, e.g.

```
while(beeperPresent())
   turnLeft();
```

- In the above code the repeated **`turnLeft()`** does not make the robot progress to having picked up all the beepers

Verifying while Loops

- while loops must be capable of working correctly no matter what the initial situations
- We can use an informal argument to check for probable correctness…

Verifying while Loops

- Show the statement works correctly when the condition is false
- Show that each time the loop body is executed, the robot's new situation is a *simpler* (less work) and *similar* (not very different) version of the old situation
- The while statement executes correctly and eventually terminates, since each execution of the loop brings the robot closer to having the condition become false

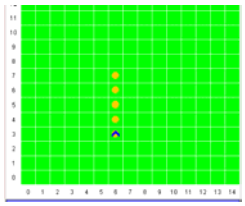
For example…

```
while(beeperPresent())
   pickBeeper();
```

- If there are no beepers in the cell, the while statement performs correctly by terminating right away.
- If there are beepers in the cell, there will be one less beeper after the instruction is executed: *Simpler* (one less beeper) and *Similar* (robot in the same cell)

Another Problem

- We want a robot to pick up a beeper and then move forward as long as there is beeper in the cell
- I.e. in the loop you will have to execute two instructions, pickBeeper() and move()



A Block of Instructions

- You can repeat a group of instructions by placing them in a block.

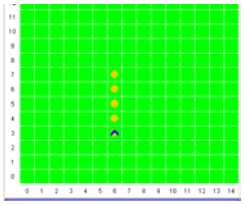
```
void harvestRow()
{
   while(beeperPresent())
   {
      pickBeeper();
      move();
   }
}
```

As with the if statement, braces can be used to create a block of instructions

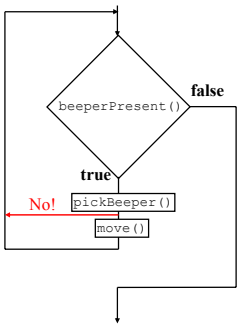
When is the condition Checked?

- Where will the robot end up?

```
void harvestRow()
{
   while(beeperPresent())
   {
      pickBeeper();
      move();
   }
}
```



When is the condition Checked?



When is the condition Checked?

- The Robot checks the condition only before it executes the body of the loop
- It is totally insensitive to the condition while it is executing instructions inside the loop body
- That is, once the robot starts executing the loop body, it does not recheck the condition until it has completely executed the loop body, and started to re-execute the entire while statement

NOT THIS WAY!

- Some beginning programmers *wrongly* think the robot checks the condition after each instruction inside the loop body
- NO!!! What would happen in the *harvestRow* method then?

```
void harvestRow()
{
   while(beeperPresent())
   {
      move();
      pickBeeper();
   }
}
```

More Examples

Write a method that will pick up beepers until a wall is encountered.

```
void harvestToWall()
{
   pickBeeper();
   while(frontIsClear())
   {
      move();
      pickBeeper();
   }
}
```

Why is the first pickBeeper() there?

What would happen if it wasn't?

How many moves and how many pickBeepers are executed?

What happens if the braces are removed?

Don't Forget the braces!

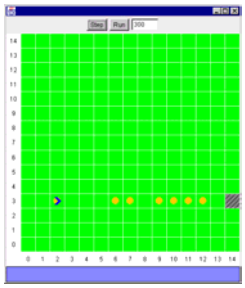
```
void harvestToWall()
{
   pickBeeper();
   while(frontIsClear())
      move();
      pickBeeper();
}
```

- In what situation does this cause an error?
- In what situation does this method work correctly?
- If you forget the braces, it may be a logic error, cause an error, or it may even work!
- Don't forget the braces!

if instructions in while loops

```
void harvestToWall()
{
   if(beeperPresent())
      pickBeeper();
   while(frontIsClear())
   {
      move();
      if(beeperPresent())
         pickBeeper();
   }
}
```

Note that we could write a pickBeeperIfPresent() method that could do this.



An Even More General Method

```
void betterHarvestToWall()
{
   clearBeepers();
   while(frontIsClear())
   {
      move();
      clearBeepers();
   }
}

void clearBeepers()
{
   while(beeperPresent())
      pickBeeper();
}
```

This method can handle any number of beepers on a cell (0, 1 or many)

General Programming advice

- Try to solve complex problems by looking for similar but simpler problems (to develop structure of solution)
- Strive to re-use your previous work

Summary

- While loops are used to repeat an instruction until some condition becomes false.
- Beware of infinite loops, check your conditions carefully.
- A while loop may be used to force some condition to become false (or true if you use the not operator).
- As with the if statement, braces may be used to create a block of instructions.