Repetition: "loops" (JFS Chap 8)

In this section:

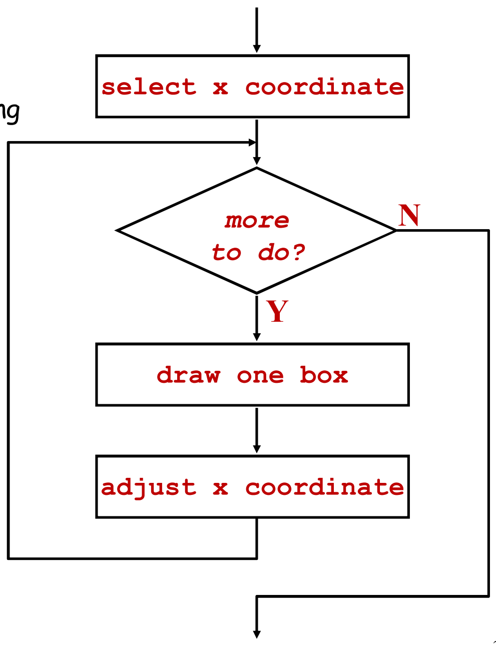
- The need to "iterate", i.e. *cause repeated execution of a section of program*
- Three Java mechanisms for this, using
  - the **while** loop
  - the **for** loop
  - the **do while** loop
- Including:
  - Random numbers

The TenBoxes Example
(variant of JFS example)

- Suppose that we need to draw a row of (say) ten boxes (small rectangles) on the screen
  - A simple, direct solution would be to use a sequence of 10 **drawRects**
- However a more flexible/adaptable approach is desirable
  - Remember that customers can be awkward!
  - For example: more or less boxes, a *variable number of boxes*, or a *column* of boxes
- As a general solution, consider the following semi-formal *"algorithm"*

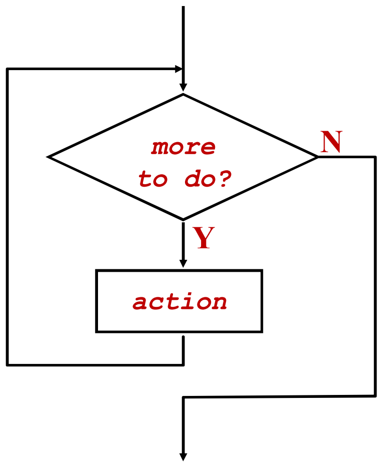
```
select x coordinate of first box;
repeat (until enough boxes have been drawn):
    draw one box;
    adjust x coordinate;
```

- We could represent this algorithm as a flow diagram involving a "loop back":



The `while` statement (JFS, p153)

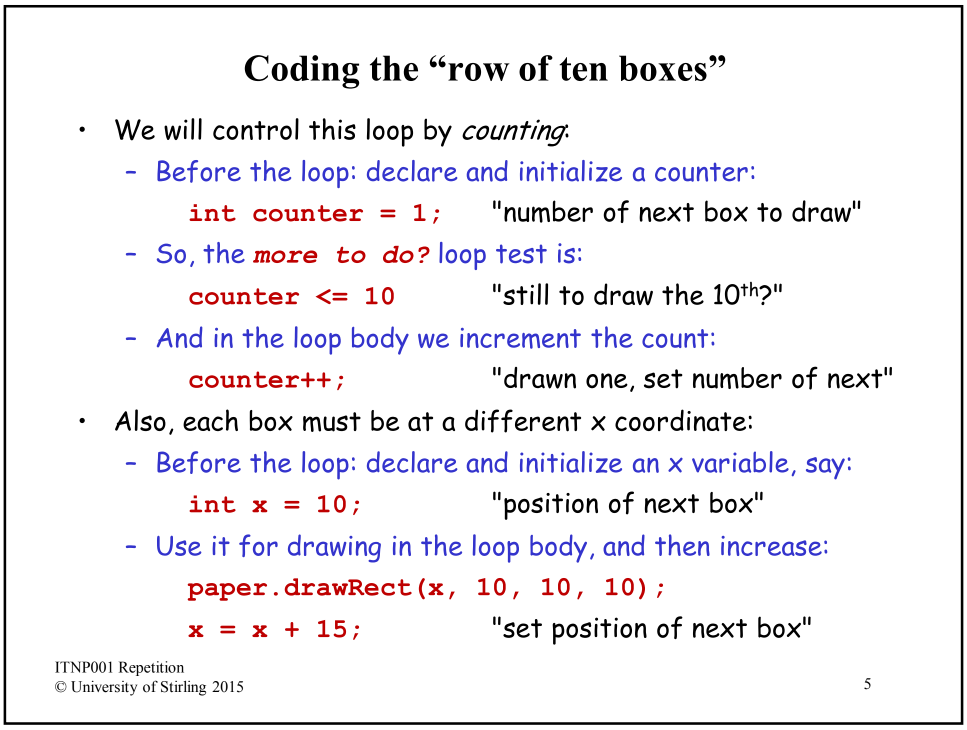
- To code "more to do?" algorithms, Java provides the `while` statement to represent the basic loop structure:



is coded as

```
while (more to do?) {
  action
}
```

- *more to do?* is a test condition (as with `if`)
- *action* is often called the "loop body"



Coding the "row of ten boxes"

- We will control this loop by *counting*:
  - Before the loop: declare and initialize a counter:

    `int counter = 1;`   "number of next box to draw"
  - So, the ***more to do?*** loop test is:

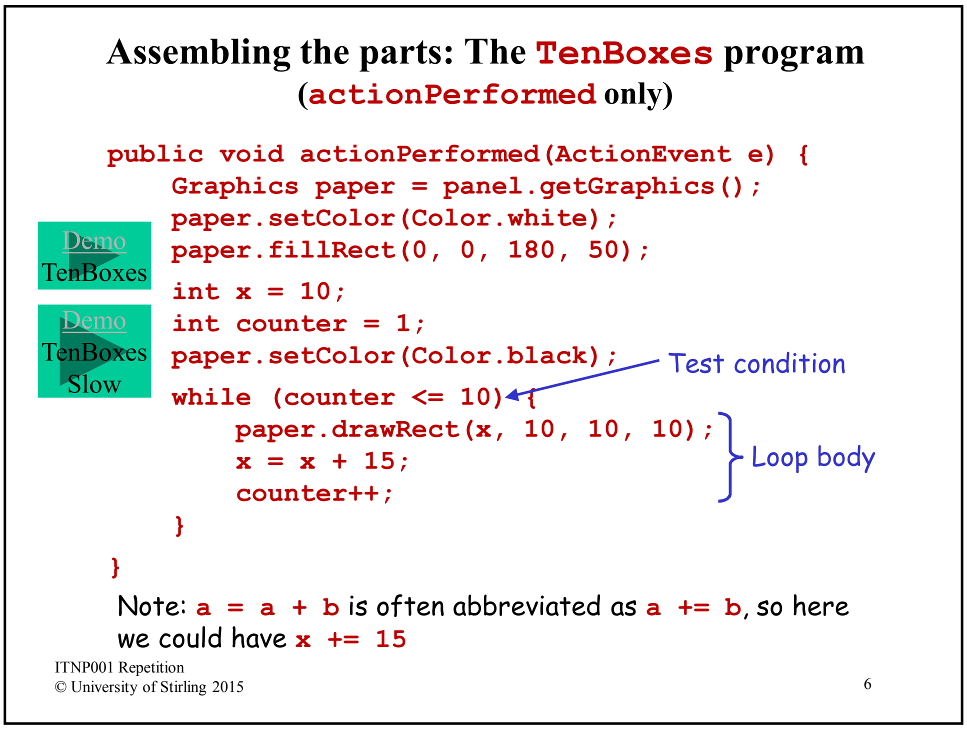
    `counter <= 10`   "still to draw the 10th?"
  - And in the loop body we increment the count:

    `counter++;`   "drawn one, set number of next"
- Also, each box must be at a different x coordinate:
  - Before the loop: declare and initialize an x variable, say:

    `int x = 10;`   "position of next box"
  - Use it for drawing in the loop body, and then increase:

    `paper.drawRect(x, 10, 10, 10);`

    `x = x + 15;`   "set position of next box"



Assembling the parts: The TenBoxes program (actionPerformed only)

Demo TenBoxes

Demo TenBoxes Slow

```
public void actionPerformed(ActionEvent e) {
    Graphics paper = panel.getGraphics();
    paper.setColor(Color.white);
    paper.fillRect(0, 0, 180, 50);
    int x = 10;
    int counter = 1;
    paper.setColor(Color.black);
    while (counter <= 10) {           Test condition
        paper.drawRect(x, 10, 10, 10);
        x = x + 15;                   Loop body
        counter++;
    }
}
```

Note: `a = a + b` is often abbreviated as `a += b`, so here we could have `x += 15`

Key points: **while** statement

- The structure of the **while** statement is:

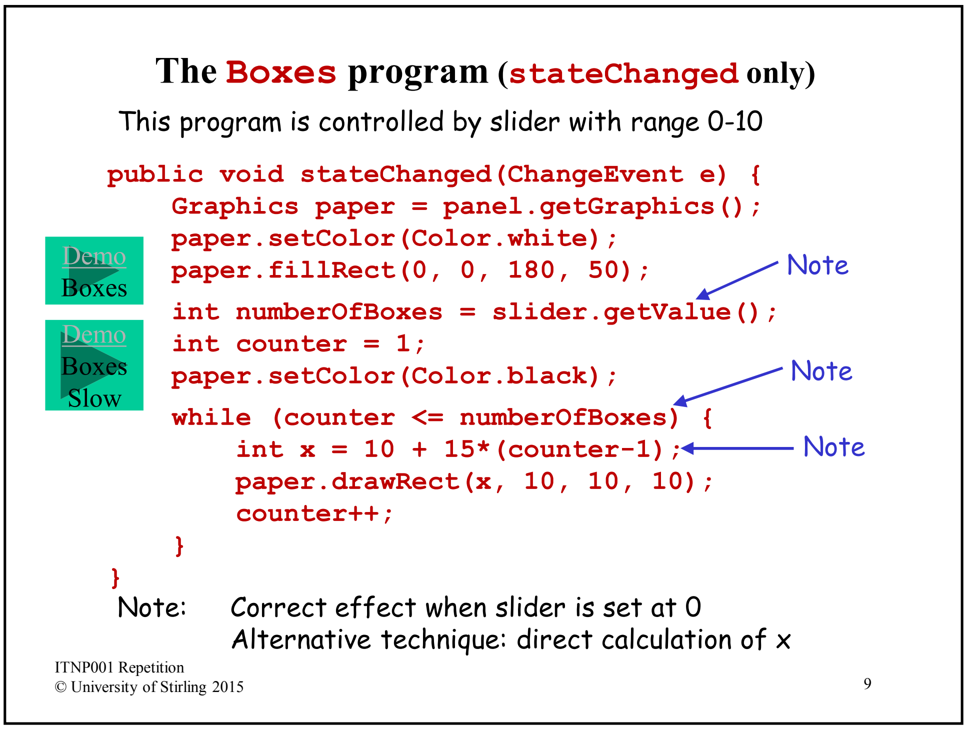
```
while ( condition ) {
  action
}
```

where, as in an **if** statement,

  - *condition* is any logical (**boolean**) expression
  - *action* can be either a single statement or a sequence (and { … } can be omitted if a single statement)

- A **while** statement may itself be a *single step* in an enclosing sequence of program statements
  - Also, may be in either action part of an **if** statement
  - ... and vice versa

- In a **while** loop, the test is performed *before each execution of the loop body*
  - In particular before the *first* execution
- So if the test evaluates to **false** the first time, then the loop body will not be executed even once!
  - This is often *exactly* the effect that we need
- Conversely, if the loop test *never* evaluates to **false**
  - The repetition will continue for ever: "infinite looping"
  - So we must make sure that execution of the loop body takes some action so that the loop test turns out **false** eventually
- It is easy to adapt this solution:
  - More or less boxes: alter the 10 in the loop test
  - Variable number: use a variable in the test (see next)
  - Column of boxes: use a variable for y instead!



The Boxes program (stateChanged only)

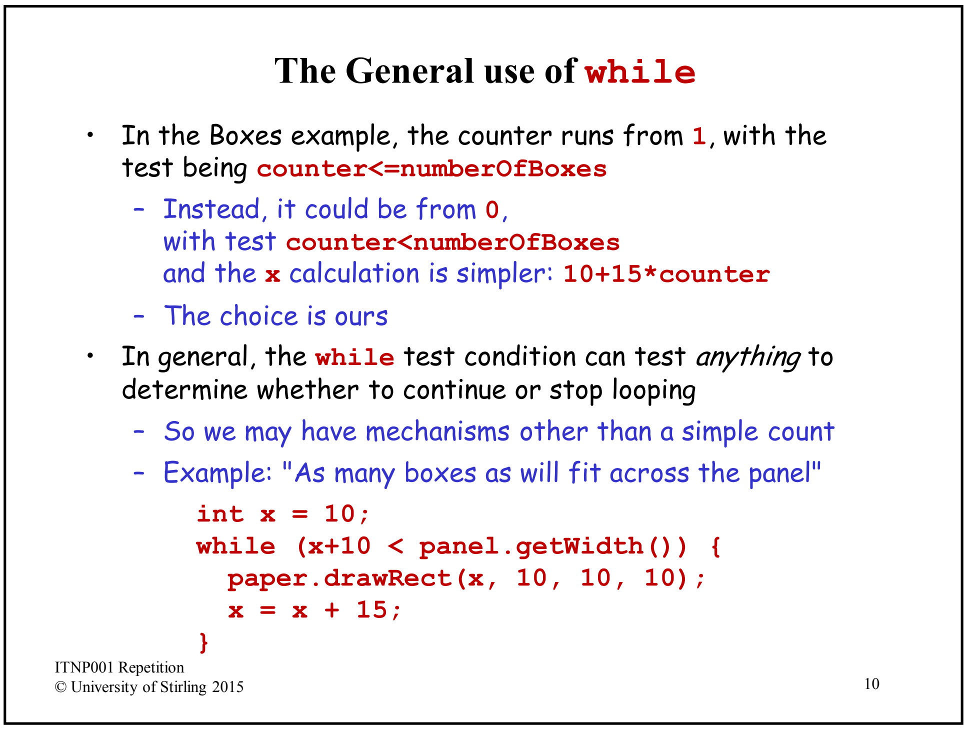
This program is controlled by slider with range 0-10

Demo Boxes

Demo Boxes Slow

```
public void stateChanged(ChangeEvent e) {
    Graphics paper = panel.getGraphics();
    paper.setColor(Color.white);
    paper.fillRect(0, 0, 180, 50);                    Note
    int numberOfBoxes = slider.getValue();
    int counter = 1;
    paper.setColor(Color.black);                      Note
    while (counter <= numberOfBoxes) {
        int x = 10 + 15*(counter-1);  ←───────────   Note
        paper.drawRect(x, 10, 10, 10);
        counter++;
    }
}
```

Note: Correct effect when slider is set at 0
Alternative technique: direct calculation of x



The General use of while

- In the Boxes example, the counter runs from 1, with the test being counter<=numberOfBoxes
  - Instead, it could be from 0, with test counter<numberOfBoxes and the x calculation is simpler: 10+15*counter
  - The choice is ours
- In general, the while test condition can test *anything* to determine whether to continue or stop looping
  - So we may have mechanisms other than a simple count
  - Example: "As many boxes as will fit across the panel"

```
int x = 10;
while (x+10 < panel.getWidth()) {
  paper.drawRect(x, 10, 10, 10);
  x = x + 15;
}
```

The `for` Statement (JFS, p158)

- The `for` statement is an alternative packaging for *some* uses of `while` statements
- The structure of the `for` statement is:

```
for (initial; test; step) {
   action
}
```

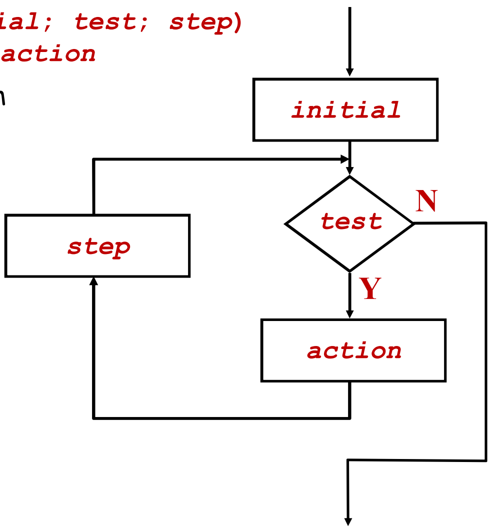
where

- `initial` and `step` are *statements*
- `test` is a `boolean` expression (as in a `while` or `if`)
- `action` is one or more statements in a block (and `{` `}` may be omitted if only one statement)

- Here is a flow diagram indicating the precise operation of

```
for (initial; test; step)
        action
```

- It is a specialized form of the `while` loop flow diagram



- A typical example:

```
for (i = 0; i < 4; i++)
  statement;
```

  - The effect of this at run time is the same as:

```
i = 0;
// i<4 is true, so:
statement;
i++;
// i<4 is true, so:
statement;
i++;
// i<4 is true, so:
statement;
i++;
// i<4 is true, so:
statement;
i++;
// i<4 is false
```

  - Often `statement;` makes use of `i`, so the values of `i` *may matter*

Example: BoxesFor: A row of boxes, again

Demo
BoxesFor

```
public void stateChanged(ChangeEvent e) {
    Graphics paper = panel.getGraphics();
    int numberOfBoxes = slider.getValue();
    paper.setColor(Color.white);
    paper.fillRect(0, 0, 180, 50);
    paper.setColor(Color.black);
    for (int counter = 1; counter <= numberOfBoxes;
                                        counter++) {
        int x = 10 + 15*(counter-1);
        paper.drawRect(x, 10, 10, 10);
    }
```

Example: SumUp (not in JFS)

- The sum of the numbers from m *up to* n, inclusive, is to be displayed, where the values of m and n are input via textfields
- First think about how it could be done manually…
- The sum is calculated by a `for` loop
- The `actionPerformed` method:

Demo
SumUp

```
public void actionPerformed(ActionEvent event) {
  int rangeStart =
               Integer.parseInt(start.getText());
  int rangeEnd = Integer.parseInt(end.getText());
  int total = 0;
  for (int count = rangeStart; count<=rangeEnd;
                                    count++)  {
    total = total+count;
  }                                  The core algorithm
  result.setText(Integer.toString(total));
}
```

Think about special cases…

for loop: extra points

- Optionally, when the counter variable is *referred to only within the loop,* we may declare the counter variable *inside the loop initialization*:

```
    for (int i = 0; i < 4; i++)
      loop body
```

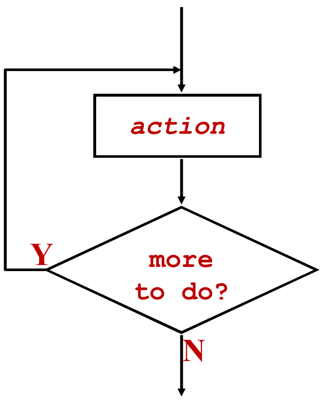
  - Both **BoxesFor** and **SumUp** use this
- Any or all of the ***initial***, ***test*** or ***step*** parts may be omitted:
  - The semi-colons *must* still be present
  - An omitted ***test*** is treated as **true** – an infinite loop!
  - *Can give obscure code – avoid!*

The do while loop (JFS p161)

- In while and for loops:
  - The test condition is checked *just before* each execution of the loop body
- Sometimes (much more rarely) what we really need is:
  - The loop body is executed and *then* the test is checked
- Example:
  - Repeatedly obtaining a user's password until the correct password is entered
  - Randomly choosing two *different* lottery numbers – see next example

The do while Statement

- To code "test after" algorithms, Java has the do while statement:

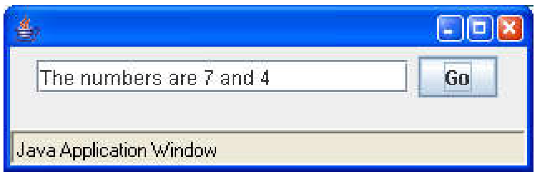


is coded as

```
do {
 action
}
while (more to do?);
```

- *more to do?* is a test condition
- *action* is the "loop body"

Example: Choosing two Lottery numbers (JFS p162)



The numbers are 7 and 4

Demo Lottery

Demo Lottery Full

- Each click on Go two *different* numbers to be chosen at random in the range 1-10
  - The first number can be *any* number from 1-10
  - The second is also a random choice from 1-10, but we must choose again if it is the same as the first - repetition
  - *There must be at least one attempt to choose the second number*
- Here is the `actionPerformed` method...

Use of do … while in the Lottery example

```
public void actionPerformed(ActionEvent event) {
  int number1, number2;
  number1 = random.nextInt(10) + 1; // Pick first
  do {
    number2 = random.nextInt(10) + 1;  // Second??
  }
  while (number1 == number2); // Repeat if duplicate
  textField.setText("The numbers are "
        + Integer.toString(number1) + " and "
        + Integer.toString(number2));
}
```

Random numbers

- A "random number generator" must be set up at the start of the program, typically:

```
import java.util.*;       // Random is in util
private Random random;    // Global variable
random = new Random();    // In createGUI
```

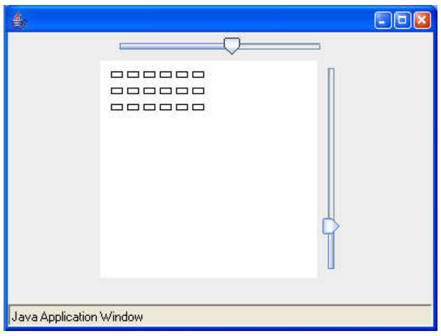
- `random.nextInt(n)` is a library method that returns a "pseudorandom" integer in the range `0` up to `n-1` (see JFS, p92)
- So `random.nextInt(10) + 1` calculates a random integer in the range 1-10
- `Random` has a variety of other methods

Choosing between loops

- We have the **`while`** and **`for`** loops, and the **`do while`** loop, available in Java, but which to choose when we need a repetition algorithm?
- The choice is a subtle one, but here are useful rules of thumb:
  - Use a **`for`** loop for a count controlled loop
  - Use a **`while`** loop for nearly all others
  - Use a **`do while`** loop *only occasionally*, instead of a **`while`** loop when appropriate - it looks tempting *but is not usually appropriate*
- Note: The **`while`** loop is strictly more general than the **`do while`** loop
  - Everything that can be programmed with a **`do while`** loop can be programmed neatly with a **`while`** loop instead
  - But not vice versa

Nested loops

- Loop statements and `if` statements are single statements
  - They may be used anywhere that a statement is expected
  - In particular a loop may have another loop statement "nested" inside it
- Here is a nested loop example from JFS (p161):
  - To draw a block of "apartments":



Demo Apartments

Slow demo Apartments

```
int y = 10;
int apartments = slider1.getValue();
int floors = slider2.getValue();          "Outer loop"
for (int floor = 1; floor <= floors; floor++) {
    x = 10;
    for (int count = 1; count <= apartments;
                                    count++) {
        paper.drawRect(x, y, 10, 5);
        x = x + 15;
    }                                 "Inner loop"
    y = y + 15;
}
```

- One "apartment" is drawn per repetition of the "inner loop" body
- The inner loop draws one complete floor:
  - A series of rectangles at differing **x** but fixed **y** coordinates
- One floor is drawn per repetition of the "outer loop" body
- The outer loop draws the entire apartment block
  - A series of floors at differing **y** coordinates

End of section