PART 6: REPEATING ACTIONS

Part 5 introduced `if` statements, which select an operation from two alternatives. This week introduces loop statements, which repeat an operation many times over. (Chapter 6 of CCJE).

- while statements and do-while statements..
- The 'loop and a half' problem.
- FOR statements.
- Nested loops.

while Statements and do-while statements

To introduce loop statements, use very simple programming problem: adding up some numbers entered from keyboard by user.

If we know how many numbers user is going to enter, can write non-loop code to read them one at a time, keeping a total of the numbers that have been read so far. Eg, for 4 numbers:

```
System.out.println("Enter the values.");
double total = 0;
total = total + in.readDouble();
total = total + in.readDouble();
total = total + in.readDouble();
total = total + in.readDouble();
System.out.println("The total is " + total);
```

Loop statements

Simple approach is only feasible for small, known, number of values.

For general case, need idea of a *loop*.

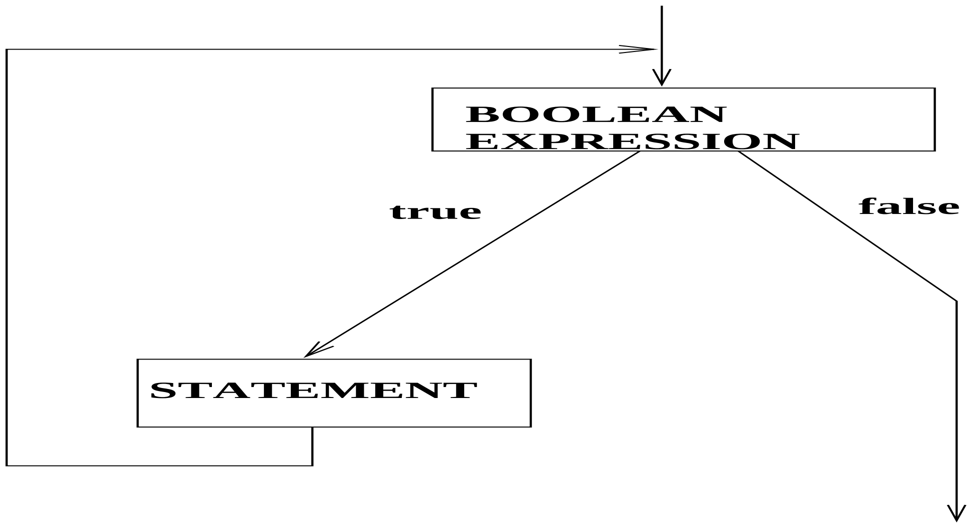
The simplest Java loop statement is the *while* statement. Has syntax:

```
while (BOOLEAN EXPRESSION)
{ STATEMENT(s) }
```

This means:

> Keep repeating the STATEMENT(s). Each time, before STATEMENT(s) is obeyed, check that the BOOLEAN EXPRESSION is true. If it is false, finish immediately.

Control flow of `while` *statement*



Using `while` *loops*

To execute a statement n times can use `while` loop with a loop counter, i.e. variable that contains count of how many times the statement has been obeyed so far. [Note: there is a better way of doing this, using a `for` loop, see later].

The variable i is the counter:

```
int i = 0;
while (i < n)
{ STATEMENT
  i++;
}
```

When body of while statement (i.e. part in curly brackets) has been executed n times, the counter will reach n, and loop will stop.

`while` *loops*

For example, here is code for finding sum of n input values.

```
double total = 0;
System.out.println("Enter the data values.");
int i = 0;
while (i < n)
{  total = total + in.readDouble();
   i++;
}
System.out.println("Total = " + total);
```

Here is complete program that adds up numbers entered by user. At start program asks user how many input values there are going to be.

EXAMPLE 6.1

```
public class Total
{ public static void main(String[] args)
  { ConsoleReader in = new ConsoleReader(System.in);
    double total = 0;

    System.out.print("How many values are there: ");
    int n = in.readInt();
    int i = 0;
    System.out.println("Enter the values.");
    while (i < n)
    { total = total + in.readDouble();
      i++;
    }
    System.out.println("Total = " + total);
  }
}
```


Euclid's Algorithm

Suppose you want to work out highest common factor of two integers, e.g. 338 and 546. (hcf is largest number that divides both of them an exact number of times.) Exhaustive search not practical.

Instead use Euclid's algorithm: Write down the two numbers, subtract smaller number from bigger one. The smaller one is unchanged. Keep repeating subtraction step until two numbers you have are equal. This value is hcf of numbers you started with.

Here is the process applied to 338 and 546:

Start:	338	546
Step 1	338	546
Step 2	338	208
Step 3	130	208
Step 4	130	78
Step 5	52	78
Step 6	52	26
Step 7	26	26

In 7 steps calculated that 26 is hcf of 338 and 546.

Euclid's Algorithm

Process can be expressed by following `while` loop. Assume a and b start with the two integers we are trying to find hcf of:

```
while (a != b)
{ if (a > b)
  { a = a-b; }
  else
  { b = b-a; }
}
```

Body of loop subtracts smaller of the two values from larger. Repeated until the numbers are equal.

[If $a > 0$ and $b > 0$ initially then they will be so at end of every iteration, so loop must eventually terminate] `do-while` *statement*

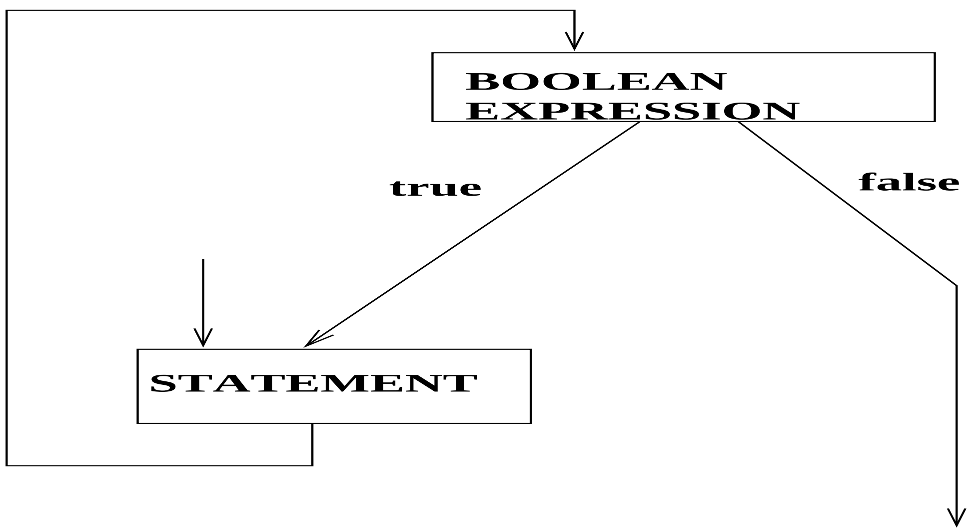
A `while` statement starts by carrying out the loop test. A `do-while` statement starts by executing body of loop, then does the test:

```
do
{ STATEMENT(s) }
while (BOOLEAN-EXPRESSION);
```

It means:

> Keep repeating STATEMENT(s). Each time, after STATEMENT(s) executed, check that BOOLEAN EXPRESSION is true. If it is false, finish immediately.

Control Flow of `do-while`



Using `do-while` *loops*

Use `do-while` if STATEMENT will always be executed at least once.

Eg, a program which tests user's mental arithmetic, repeatedly asking user to multiply pairs of numbers together.

Here is a sketch of algorithm of program:

```
Repeat
{  Choose two numbers, a and b,
   between 1 and 12.

   Give the user the problem:
   what is a times b?

   Ask the user to input ‘yes’
   if they want another question,
   or ‘no’ if they want to finish;
```



```
   Finish unless the user’s answer is ‘yes’.
}
```

The test whether to repeat the process and set another problem comes at the end. So this is an obvious candidate for a do-while statement. Here is a partially coded version of the process.

```
do
{  Choose two numbers, a and b,
   between 1 and 12.

   Give the user the problem:
   what is a times b?

   System.out.println("Do you want to go on? (yes/no)");
   reply = in.readLine();
} while (reply.equals("yes"));
```

Java provides objects that can be used for generating pseudo-random numbers. They belong to class *Random*. First you must create one. E.g.

```
Random rand = new Random();
```

To generate random number in the range 0 to n-1, use expression

```
rand.nextInt(n)
```

For example to set a and b to two random numbers in the range 1 to 12, do:

```
int a = rand.nextInt(12) + 1;
int b = rand.nextInt(12) + 1;
```

Every time $rand.nextInt(12)$ is evaluated, another number will be returned.

Here is multiplication program with a *Random* object used to generate the pairs of numbers. *Random* class is in the `java.util` package in Java library.

EXAMPLE 6.2

```
import java.util.*;

public class Multiply
{ /* Test if user knows her multiplication tables. */
  public static void main(String[] args)
  { ConsoleReader in =
        new ConsoleReader(System.in);
    Random rand = new Random();

    System.out.println("Here is a test for you.");
    String reply;
    do
    { /* Choose two numbers between 1 and 12. */
      int a = rand.nextInt(12) + 1;
      int b = rand.nextInt(12) + 1;
```

```
      /* Test if the user knows their product. */
      System.out.println("What is " + a +
                         " times " + b + "?");
      int answer = in.readInt();
      if (answer == a*b)
      { System.out.println("Well done!"); }
      else
      { System.out.println("No. The answer is " + a*b); }
      /* Check if user wants to go on. */
      System.out.println("Do you want another test (yes/no)?");
      reply = in.readLine();
    }
    while (reply.equals("yes"));

    System.out.println("Bye");
  }
}
```


The 'Loop and a Half' Problem

- Frequent problem – a set of statements to be repeated in loop, and you know what boolean expression to use to test if time has come to finish. But natural place for test is in middle of statement being repeated.
- If test could be done at beginning, would use a `while`.
- If at end, would use `do-while` statement.

Several different solutions.

Typical example of problem

Suppose want to write program that computes square roots for user. Program asks user to enter a number x and, provided $x \geq 0$, will display square root of x. If user enters a negative number, program terminates:

```
Repeat
{  Read the next input value
   and store it in x;
   If x is negative then finish;
   Output the square root of x;
}
```

Test is after x has been read, and before square root is computed.

Following are three possible ways to turn this process into Java code.

Version 1

Read x first time before starting loop. Do `while` loop that tests if $x \geq 0$:

```
Read the next input value
and store it in x;
while (x >= 0)
{  Output the square root of x;
   Read the next input value
   and store it in x;
}
```

This works, but involves writing statement that reads x twice.

Problem for maintenance especially if duplicated code was larger.

Version 2

This version uses a `break` statement. This statement says: finish innermost loop or switch statement being executed at the moment. [See Winder & Roberts, 2nd Edition, page 824]

In this version, `break` statement terminates loop, not the Boolean expression at start of `while` statement (which is always true).

Using a loop that starts with `while (true)`, together with a `break` statement, gives the pseudocode:

```
while (true)
{  Read the next input value
   and store it in x.
   if (x < 0) { break; }
   Output the square root of x.
}
```


Version 3

This uses `while` statement together with Boolean variable *done*, set to *false* at start, is *true* when program should terminate. Such a variable is called a 'flag'.

```
boolean done = false;
while (! done)
{  Read the next input value
   and store it in x;
   if (x >= 0)
   { Output the square root of x; }
   else
   { done = true; }
}
```

Logically correct, but seems excessive code for this simple problem.

EXAMPLE 6.3

```
public class SqrtLoop
{ /* Read floating-point numbers entered by user
     and output their square roots.
     Finish when user enters a value < 0.
   */
   public static void main(String[] args)
   { ConsoleReader in =
         new ConsoleReader(System.in);
     System.out.println("Square root program.\n" +
                        "(Input value < 0 to finish.)");

     while (true)
     { System.out.print("Enter number: ");
       double x = in.readDouble();
       if (x < 0)
       { break; }
```



```
       System.out.println("Square root = " + Math.sqrt(x));
     }
     System.out.println("Bye");
  }
}
```

Could put loop inside a separate method and use `return` instead of `break` – may be clearer.

Alternative – explicit quit

```
while (true)
{  System.out.print("Enter number or quit: ");
   String reply = in.readLine();
   if (reply.equalsIgnoreCase("quit"))
   { break; }
   double x = Double.parseDouble(reply);
   System.out.println("Square root = " + Math.sqrt(x));
}
```


for loops

Used whenever a pre-determined number of iterations is needed (instead of iterating until particular input value received).

Approach using `while`:

```
Set variable to first value.
while (variable has not passed final value)
{  STATEMENTS
   Set variable to next value.
}
```

Eg, display table of values of $cos(x)$ for x being: 0, 5, 10, 15, …, 85, 90:

```
double x = 0;
while (x <= 90)
{  Display the value of x and cos(x).
   x = x+5;
}
```

`for` *loop*

A loop of this form should be expressed using a `for` loop which has general form:

```
for (First; Continue; Next)
{ STATEMENTS }
```

First is a statement that sets index variable of loop to its first value. *Continue* is boolean expression, tests if variable has not passed final value. *Next* is statement defining how index variable is updated.

Using `for`, get:

```
for (double x = 0; x <= 90; x = x + 5)
{ System.out.println("\t" + x + "\t" + Math.cos(x)); }
```

[Never use == as termination test for *double* indexes!]

`for` *loops*

Simplest example `for` loop is repeating a statement n times. `while` version was:

```
int i = 0;
while (i < n)
{  STATEMENT;
   i++;
}
```

`for` version is:

```
for (int i = 0; i < n; i++)
{ STATEMENT; }
```

Very common form of loop – always use `for` in this case to make code easier to understand.

Applet example

Applet that paints number of coloured circles in its display area. Positions, sizes and colours of circles chosen randomly. Pseudocode:

```
for (int i = 0; i < HOW_MANY; i++)
{ Choose a random circle and a random colour
  and paint the circle with the colour.
}
```

Everything done in *paint* method.

If width of display area is w, height h, random circle produced by choosing following random values.

1. An x-coordinate in range 0 to w.
2. A y-coordinate in range 0 to h.
3. A radius in range 1 to MAX_RADIUS.

The constant MAX_RADIUS is an upper bound on circle radius.

```
int centx = random.nextInt(w+1);
int centy = random.nextInt(h+1);
int rad = random.nextInt(MAX_RADIUS) + 1;
Shape circle =
   new Ellipse2D.Double(centx-rad, centy-rad,
                        2*rad, 2*rad);
```

To generate random colour, three random float values in range 0 to 1 are used for red, green and blue components of colour.

The method call

```
random.nextFloat()
```

returns random float value in range 0 to 1.

So following code will produce required colour.

```
float red = random.nextFloat();
float green = random.nextFloat();
float blue = random.nextFloat();
Color color = new Color(red, green, blue);
```


Applet example

Then render the circle:

```
g2.setColor(color);
g2.fill(circle);
```

Here is complete applet.

EXAMPLE 6.4

```
import java.applet.*;
import java.awt.*;
import java.awt.geom.*;
import java.util.*;

/* Display HOW_MANY random circles.
   The size of the circles is
   limited by MAX_RADIUS. */
```

```
public class Bubbles extends Applet
{  private final int HOW_MANY = 100;
   private final int MAX_RADIUS = 50;

   public void paint(Graphics g)
   {  Graphics2D g2 = (Graphics2D) g;
      Random random = new Random();
      int w = getWidth();
      int h = getHeight();

      for (int i = 0; i < HOW_MANY; i++)
      {  /* Choose a random circle. */
         int centx = random.nextInt(w+1);
         int centy = random.nextInt(h+1);
         int rad = random.nextInt(MAX_RADIUS) + 1;
         Shape circle =
            new Ellipse2D.Double(centx-rad, centy-rad,
                                 2*rad, 2*rad);
```



```
         /* Choose a random colour. */
         float red = random.nextFloat();
         float green = random.nextFloat();
         float blue = random.nextFloat();
         Color color = new Color(red, green, blue);

         /* Paint the circle with
            the colour. */
         g2.setColor(color);
         g2.fill(circle);
      }
   }
}
```

To show particular set of circles, create random number generator with:

```
Random random = new Random(SEED);
```

where constant SEED is a value of type `long`. This value will set random number generator so it generates a particular sequence of values.

If seed is missed out, as in program above, Java will use seed produced by evaluating

```
System.currentTimeMillis()
```

This is time elapsed since since midnight, 1 January 1970, measured in milliseconds.

Graph display example

This program plots graphs of functions $y = f(x)$.

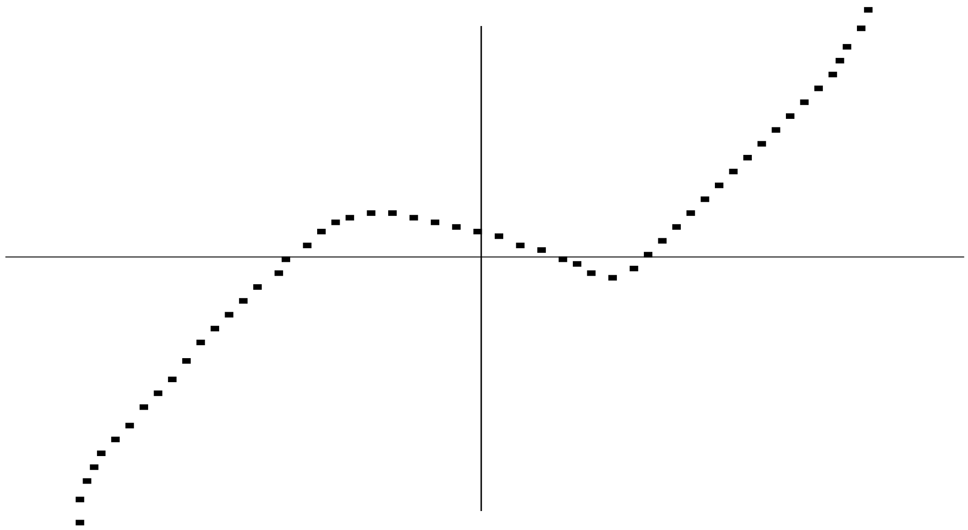
Function f is represented by method (called f) that, given a parameter value x returns corresponding value of the function. For example, for the graph of

$$y \;=\; x^3/100 - x + 2$$

the following method was used.

```
private static double f(double x)
{  return x*x*x/100 - x + 2; }
```

Graph for cubic function



Graph display example

Axes of graph are in middle of display area. Values of x range from $-xLimit$ to $+xLimit$, $xLimit$ is calculated using width of display area. Distance between two successive x values is denoted by XSTEP.

Pseudocode is:

```
for (double x = -xLimit; x <= xLimit; x = x+XSTEP)
{ Compute y = f(x)
  and plot the point (x,y).
}
```

Coordinates

Will assume that 1 unit distance on axes is equal to SCALE pixels, where SCALE is constant set to suit whatever function is being plotted.

Suppose a point has coordinates (gx,gy) measured on the graph axes, and coordinates (dx,dy) measured on the display axes. Then:

```
dx  =  dw/2 + gx*SCALE
```

where *dw* is width of display area.

Similarly, if height of display area is *dh* pixels, we have

```
dy  =  dh/2 - gy*SCALE
```


Main loop has form:

```
for (double gx = -gxLimit;
     gx <= gxLimit;
     gx = gx + GXSTEP)
{ Calculate y = f(x).
  Convert (gx,gy) to display coordinates (dx,dy).
  Plot the point (dx,dy) in the display area.
}
```

Plotting point (dx, dy) is done by filling square defined by

```
new Rectangle(dx, dy, 1, 1)
```

Unfortunately, Java does not allow us to plot a single pixel in an applet display area.

EXAMPLE 6.5

```
import java.applet.*;
import java.awt.*;
import java.awt.geom.*;

public class Graph1b extends Applet
{  /* Display graph of y = f(x) for some function f.
      The origin of the graph axes is
      at the centre of the display area.
      SCALE = size in pixels of 1 unit
              on the axes.
      GXSTEP = difference between one x
              value and the next on
              the graph.
   */

   private final double SCALE = 12;
```



```
   private final double GXSTEP = 0.25;

   public void paint(Graphics g)
   {  Graphics2D g2 = (Graphics2D) g;
      double dw = getWidth();
      double dh = getHeight();

      /* Draw the axes. */
      g2.setColor(Color.red);
      g2.draw(new Line2D.Double(0, 0.5*dh, dw, 0.5*dh));
      g2.draw(new Line2D.Double(0.5*dw, 0, 0.5*dw, dh));

      /* Plot the graph. */
      g2.setColor(Color.black);
      double gxLimit = 0.5*dw/SCALE;
      for (double gx = -gxLimit; gx <= gxLimit;
           gx = gx+GXSTEP)
      {  double gy = f(gx);
```

```
        /* Convert gx and gy to display
           coordinates dx and dy. */
        int dx = (int) Math.round(0.5*dw + gx*SCALE);
        int dy = (int) Math.round(0.5*dh - gy*SCALE);

        /* Plot the point (dx,dy). */
        g2.draw(new Rectangle(dx, dy, 1, 1));
     }
  }

  private static double f(double x)
  {  return x*x*x/100 - x + 2; }
}
```


In a `for` statement such as

```
for (int i = 0; i < n; i++)
{ STATEMENT }
```

Because of the declaration

```
int i = 0;
```

variable i is created as soon as Java starts to execute loop. i can be used anywhere inside STATEMENT (but should not be assigned to).

But variable will be deleted as soon as Java finishes the loop. So cannot find what was final value of i.

To avoid problem, declare i before start of loop:

```
int i;
for (i = 0; i < n; i++)
{ STATEMENT }
```

Nested Loops

Example 2 was program that set multiplication problems for user. It was based on following outline:

```
Repeat
{  Choose two numbers, a and b,
   between 1 and 12.

   Give the user the problem:
   what is a times b?

   Ask the user to input ‘yes’
   if they want another question,
   or ‘no’ if they want to finish;

   Finish unless the user’s answer is ‘yes’.
}
```


We replaced Repeat by `do-while` statement, and actions in body of loop were converted into code independently. This can still be done even if we make one of actions more elaborate so it requires a loop of its own.

Instead of one attempt for each product, give user as many attempts as they want. So need to change part of program that was specified by pseudo-code:

```
Give the user the problem:
what is a times b?
```

If user can have many attempts at answering problem, need a loop instead.

Here is a sketch of how it might look:

```
Repeat
   Ask the user what is a times b.
   Read the user's answer.
   If user says 'I give up',
   give the correct answer and finish this loop.
   If they give the correct answer,
   say 'Well done' and finish this loop.
   Say 'No. Try again'.
```

Here is code for this part of program. It is a `while (true)` loop with two `break` statements:

```
while (true)
{ System.out.println("What is " + a + " times " + b + "?");
  reply = in.readLine();
  if (reply.equalsIgnoreCase("Give up"))
  { System.out.println("The answer is " + a*b);
    break;
```



```
  }

  int value = Integer.parseInt(reply);
  if (value == a*b)
  { System.out.println("Well done!");
    break;
  }
  System.out.println("No. Try again.");
}
```

Coding up rest of outer loop gives:

```
do
{ /* Choose two numbers between 1 and 12. */
  int a = random.nextInt(12) + 1;
  int b = random.nextInt(12) + 1;

  Give the user the problem:
  what is a times b?

  /* Check if user wants another question. */
  System.out.println("Do you want to go on (yes/no)?");
  reply = in.readLine();
} while (reply.equalsIgnoreCase("yes"));
```

Combining the two, get following complete program, which has `while` loop nested inside a `do-while`. Notice how indentation of code is used to show structure of the program.

EXAMPLE 6.6

```
import java.util.Random;

public class Multiply2
{ /* Test if user knows her multiplication tables.
   */
  public static void main(String[] args)
  { ConsoleReader in =
      new ConsoleReader(System.in);
    Random random = new Random();

    System.out.println("I am going to give you a test.");
    String reply;
    do
    { /* Choose two numbers between 1 and 12. */
      int a = random.nextInt(12) + 1;
      int b = random.nextInt(12) + 1;
```

```
      /* Test if the user knows their product. */
      while (true)
      { System.out.println
                ("What is " + a + " times " + b + "?");
        reply = in.readLine();
        if (reply.equalsIgnoreCase("Give up"))
        { System.out.println("The answer is " + a*b);
          break;
        }
        int value = Integer.parseInt(reply);
        if (value == a*b)
        { System.out.println("Well done!");
          break;
        }
        System.out.println("No. Try again.");
      }

      /* Check if user wants another question. */
```



```
      System.out.println("Do you want to go on  (yes/no)?");
      reply = in.readLine();
    } while (reply.equalsIgnoreCase("yes"));

    System.out.println("Bye");
  }
}
```

The following is what might appear on the screen when the program is run.

```
I am going to give you a test.
What is 1 times 9?
 9
Well done!
Do you want to go on (yes/no)?
 yes
What is 12 times 3?
 30
No. Try again.
What is 12 times 3?
 27
No. Try again.
What is 12 times 3?
give up
The answer is 36
```



```
Do you want to go on (yes/no)?
no
Bye
```

(How would you adapt this program so it only chooses numbers $a, b > 5$?)

Coursework 4. Hand-in: Friday 15th November, 4pm. Coursework office

1 Redo question 1, coursework 2, using a suitable loop to calculate and print out the successive values of $n!$ for $n = 1, \ldots, 5$.

2 Redo question 2, coursework 2, using a suitable loop to calculate and print out the successive values of $n!/2^n$ for $n = 1, \ldots, 5$.

Hand in: printouts of source code. Put your name and tutors name on your solution.

PART 7: MORE ABOUT CLASSES

This part gives a more complete picture of classes and introduces the technique of recursion. (Chapter 7 of Horstmann).

- About variables and methods.
- Interfaces.
- Recursion.