Repetition - repeating a set of actions a number of times.

- Sum $1 + 2 + 3 + 4 + 5 + \ldots + 50$
- Find $n! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times \ldots \times n$
- $m^n = m \times m \times m \ldots \times n$-times
- Reading all the numbers until user enters a special value
- Adding input numbers until sum exceeds set value

Number of times known in advance
- Definite

Number of times to repeat cannot be determined
- Indefinite
Finding average of 3 marks input

CALCULATION OF ASSIGNMENT AVERAGE
Enter mark of assignment 1: 5
Enter mark of assignment 2: 8
Enter mark of assignment 3: 10

The average mark is: 7.66

Design

calculate the sum of the marks
process the average

calculate the sum of marks
for all assignments
prompt for mark
get mark
add to previous total
process the average
calculate the average
display the average
import java.io.*;
public class AverageMarks {
    public static void main (String[] args)
        throws IOException {
        BufferedReader stdin = new BufferedReader
            (new InputStreamReader (System.in));
        String inString;
        int i, num, sum = 0;
        for (i = 1; i <= 3; i++) {
            System.out.println("Enter mark of assign" + i + " : ");
            inString = stdin.readLine();
            num = Integer.parseInt (inString);
            sum += num;  // accumulate total
        }
        System.out.println("The average is: " + (double)sum / 3);
    }
}
Behavior of the loop

1. if false exit loop else execute statement
2. for (initial; test; increment)
3. statement(s)
4. then back to 2

What is the output? (if any)

1. for (int i=3; i<3; i--)
   System.out.println("i is now " + i);

2. for (int i=3; i>0; i--)
   System.out.println("i is now " + i);

3. for (int i=0; i<3; i++)
   System.out.println(i*10);

4. for (int i=0; i<3; )
   System.out.println("i is now " + i);
Attempting to sum 1 to 10. What’s wrong?

```java
int sum = 0;
for (int i=1; i<=10; i++) {
    System.out.println("i is now "+ i);
    sum = sum + i;
    System.out.println("1 + 2..10 is "+ sum);
}
```

Ans 1. Braces missing  2. Semicolon (;) completes loop
A Conversion Table

• from cubic inches to cubic centimeters
• from 20 cubic inches down to 2, in step of 2

double cubicCm;
int finish, step;
// constant conversion factor C.I to C.C
final double CONV_FACTOR = 1000.0/61.0;

finish = 10; step = 2;
System.out.println("TABLE C.I TO C.C");
for ( int i = finish; i > 0 ; i--)  {
  cubicCm = (i * step) * CONV_FACTOR;
  System.out.println(cubicCm);
}

import java.text.*;
...
DecimalFormat df = newDecimalFormat("0.00");
System.out.println(df.format(cubicCm));
Nesting an if within for loop

Can I print all the numbers not divisible by 3 in the range 1 to 20 using a for loop?

```
for (i=1; i<=20; i++) {
    if (i%3 != 0)
        System.out.println();
}
```

Nested Loop

What will be the output of the program below?

```
for (i=1; i<=4; i++) {
    for (j=1; j<=3; j++)
        System.out.print("i+j");
    System.out.println();
}
```
Indefinite repetition

- Asked to write a program to find the average marks.
- Number of students in class may vary
- We must read the marks input until user enters -1
- A while loop is more appropriate

```java
Initialization; // if any
while (boolean_expression) {
    statement;
}
```

Design

Set sum to 0
Display Prompt “enter marks (-1) to terminate”
read mark
while (mark != -1) {
    add mark to sum
    increment number_of_students
    read next mark
}
if (number_of_students > 0) compute and print average
otherwise print error message
import java.io.*;
public class FindAverage {
    public static void main (String[] args)
        throws IOException {
            ConsoleReader console = new ConsoleReader(System.in);
            int mark, sum = 0, num = 0;
            System.out.println("Enter marks (-1) to terminate");
            mark = console.readInt();
            while ( mark != -1) {
                sum += mark;
                num++;
                mark = console.readInt();
            }
            if (num ==0)
                System.out.println("No students in class");
            else
                System.out.println("Aver = " +(double)sum/num);
        }
}

Write a program to accept characters as input, and quit when a ‘Q’ or a ‘q’ is typed, printing how many characters other than ‘q’ or ‘Q’ have been typed.

This program counts the number of characters entered.
Input a character, ‘Q’ or ‘q’ to quit: 
x
Input a character, ‘Q’ or ‘q’ to quit: 
e
Input a character, ‘Q’ or ‘q’ to quit: 
s
Input a character, ‘Q’ or ‘q’ to quit: 
p
Input a character, ‘Q’ or ‘q’ to quit: 
J
Input a character, ‘Q’ or ‘q’ to quit: 
q
The number of characters entered is: 5
Design

initialisation
show initial prompts
while input not a q or a Q
process character
prompt for a character
get character
add 1 to total
display total

import java.io.*;
public class CharTest {
    public static void main (String[] args)
        throws IOException {
            BufferedReader charInput = new BufferedReader
                (new InputStreamReader (System.in));
            char answer;
            int charTotal = 0;
            System.out.println("This program counts the "
                + "number of characters entered.");
            System.out.println("Input char 'Q' , 'q' to quit: ");
            answer = charInput.readLine().charAt(0);
            while (answer != 'Q' && answer != 'q')  {
                System.out.println("Input char 'Q', 'q' to quit:");
                answer = charInput.readLine().charAt(0);}
            charTotal++;
            System.out.println("Num. of chars = " + charTotal);
    }
}
Using a do - while loop

We could avoid the double prompting using a do-while loop and break.

do  {
    System.out.println("Input a character, "
        + "'Q' or 'q' to quit:");
    answer = charInput.readLine().charAt(0);
    if (answer == 'Q' || answer == 'q') break;
    charTotal++;
} while (true);

The general form of do-while

do  {
    statements;
} while (condition);

• Difference between while and do -while
• while is skipped altogether if the condition is false – hence it it termed ‘0 or more repetitions’
• The do loop is executed at least once, so it is called a ’1 or more repetitions’ loop.
The **break**

**break** statement allows you to break out of a loop or switch statement

```java
int i;
for (i=1; i<=10; i++) {
    if (i % 3 == 0)
        break;
    System.out.println(i);
}
System.out.println("Outside loop i is " + i);
```

---

**continue** statement

**continue** statement causes the flow of control to pass to the next iteration of the loop.

```java
int i;
for (i=1; i<=10; i++) {
    if (i % 3 == 0)
        continue;
    System.out.println(i);
}
System.out.println("Outside loop i is " + i);
```

**Coding Guideine:** Try and avoid **break** and **continue**
Program Testing

• Testing is intended to reveal any bugs.
• Debugging is intended to locate and remove them.
• Testing is an important part of program development. It should be planned from the outset.
• Program testing can never prove the absence of bugs, but it can show their presence. (Dijkstra)
• Object oriented and modular programming facilitate testing since programs are constructed with tested building blocks (classes)

Exhaustive testing

Consider a simple example of adding two integers (X+Y).
• We need to consider all possible input combinations.
• How long will it take to test all combinations of X + Y?
• Java the maximum int is $2^{31} - 1 = 2,147,483,647$
• Possible combinations are in billions taking years to complete the test
Black Box Testing

**Black-box testing:**
- does *not* consider program’s design structure
- test plan derived from program specification
- plans some specific inputs and inspect the outputs
- The test values can be reused in future

---

White-box (Glass-box) testing:

- inputs chosen to exercise (all) control paths.
- exercises them with boundary value inputs
  \[ X = a \times b / (c \times d) \]
### Debugging - locating

- A basic technique to test a piece of code is to insert *diagnostic* output statement (print) after each method call.

- When the error is seen in one output but not the one prior to it, it must be between the relevant two diagnostics.

- Then we can step into the last method call trying to locate the problem. Debuggers (JBuilder, kawa) automate this process.

---

### Input Validation

*Input validation* is the process of writing extra code to ensure the user inputs only valid data.

A simple validation technique is to:

- declare input variable of an appropriate type
- prompt user for input in the valid sub-range
- check if input value is of appropriate type and within valid sub-range
- if so, continue normal processing,
- if not, re-prompt and re-check until valid
Sample

```java
int inputDigit;

...........

do {
    // until input is a digit
    System.out.println("Please enter a digit: ");
    inString = stdin.readLine();
    inputDigit = Integer.parseInt(inString);
} while(inputDigit < 0 || inputDigit > 9);
```