Chapter 7
Repetition Structures

The while, do-while, and for statements
Program structures

- All algorithms (programs) can be expressed in terms of three structures
  - sequential structures
  - conditional structures
    - if-statements
  - repetition structures
    - while loop and other equivalent forms
Pseudo-code while loop

\textbf{WHILE} \textit{BooleanExpression} \textbf{DO} \textit{Statements} \textbf{END WHILE}

If \textit{BooleanExpression} is true the statements in the loop are executed
While statement in Java

while (BooleanExpression) {
  Statements
}

Execute these statements if BooleanExpression is true

Here BooleanExpression should eventually become false or the loop will be infinite
Flowchart for while statement

Expression

Statements

false

ture

The loop statements will never be executed if the expression is initially false
Counting up with while loop

```java
int count = 1;
while (count <= 10)
{
    System.out.print(count + " ");
    count = count + 1; // or count++;
}
System.out.println();
```

The result displayed is  1 2 3 4 5 6 7 8 9 10

Try it using the BeanShell workspace:
Choose "Capture System in/out/err" from File menu
Counting down with while loop

```java
int count = 10;
while (count >= 1)
{
    System.out.print(count + " ");
    count = count - 1; // or count++;
}
System.out.println();
```

The result displayed is  10 9 8 7 6 5 4 3 2 1

Try it using the BeanShell workspace:
Choose "Capture System in/out/err" from File menu
The 3k+1 numbers (Collatz)

- If k is even divide by two else multiply by 3 and add 1. Does the sequence end in 1?
  - k = 8
    - 8,4,2,1
  - k = 7
    - 7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1
  - k = 27
    - 112 numbers are displayed ending in 1
A loop that may not terminate

```java
long k = n;
System.out.print(k);
while (k > 1)
{
  if (k % 2 == 1) // k is odd
  {
    if (k > 3074457345618258602L)
      System.out.println("Overflow");
    k = 3*k + 1
  }
  else // k is even
  {
    k = k / 2;
    System.out.print("," + k);
  }
}
int k = 0;
while (k <= 9)
{
    double y = 10.0 + 20.0*k;
    g2D.draw(new Line2D.Double(10,y,200,y));
    k = k + 1; // k++
}
Java has some static methods to do this:
- `public int Integer.parseInt(String s)`
- `public long Long.parseLong(String s)`
- `public float Float.parseFloat(String s)`
- `public double Double.parseDouble(String s)`

The classes here are called wrapper classes.

Each method throws a `NumberFormatException` exception if the string `s` does not represent a valid number.
Algorithm (1)

- A numeric string $s$ is a sequence of digit characters: $s = c_0c_1 \cdots c_{n-1}$
- Convert each digit character to an integer in the range 0 to 9 using the formula
  \[d_k = c_k - '0'\]
- Use the formula to do the conversion
  \[d = d_{n-1}10^{n-1} + \cdots + d_110^1 + d_0\]
  \[d = d_{n-1} + 10(d_{n-1} + \cdots + 10(d_1 + 10d_0)\cdots)\]
ALGORITHM (2)

ALGORITHM $stringToInt(c_0, c_1, \ldots, c_{n-1})$

$value \leftarrow 0$

$k \leftarrow 0$

WHILE $k < n$ DO

\[
value \leftarrow (c_k - '0') + 10 \times value
\]

$k \leftarrow k + 1$

END WHILE

RETURN $value$
public class StringToIntConverter
{
    public int stringToInt(String s)
    {
        int numDigits = s.length();
        int value = k = 0;
        while (k < numDigits)
        {
            value = (s.charAt(k) - '0') + 10*value;
            k = k + 1;
        }
        return value;
    }
}
import java.util.Scanner;
public class StringToIntRunner
{
    public void run()
    {
        Scanner input = new Scanner(System.in);
        StringToIntConverter converter =
            new StringToIntConverter();
        System.out.println("Enter digit string");
        String digitString = input.nextLine();
        int value = converter.stringToInt(digitString);
        System.out.println("int value is " + value);
    }

    public static void main(String[] args)
    {
        new StringToIntRunner().run();
    }
}
Square root algorithm

To find $\sqrt{a}$ start with $x_0 = a / 2$

Compute the terms in the sequence

$$x_n = \frac{1}{2} \left( x_{n-1} + \frac{a}{x_{n-1}} \right)$$
Example

\[ a = 2 \]

\[ x_0 = \frac{a}{2} = 1 \]

\[ x_1 = \frac{1}{2} \left( 1 + \frac{2}{1} \right) = 1.5 \]

\[ x_2 = 0.5(1.5 + \frac{2}{1.5}) \approx 1.46667 \]

\[ x_3 \approx 1.41422 \]

Exact answer \( \approx 1.41421356237310 \)
public double squareRoot(double a) {
    double xOld = 1;
    double xNew = a;
    while (Math.abs((xNew - xOld) / xNew) > 1E-16) {
        xOld = xNew;
        xNew = 0.5 * (xOld + a / xOld);
    }
    return xNew;
}

Here we use the relative error to decide when to stop the loop based on comparing one approximation xNew with the previous approximation xOld.
public class SquareRootCalculator
{
    public double squareRoot(double a)
    {
        ...
    }
}

put the squareRoot method in this class
import java.util.Scanner
public class SquareRootRunner
{
    public void run()
    {
        Scanner input = new Scanner(System.in);
        SquareRootCalculator calculator =
            new SquareRootCalculator();
        System.out.println("Enter number");
        double a = input.nextDouble();
        input.nextLine(); // eat end of line
        double root = calculator.squareRoot(a);
        System.out.println("Square root of " + a + " is " + root);
    }
}
SquareRootRunner (2)

```java
System.out.println("Square of root is " + root * root);
System.out.println("Square root using " + " Math.sqrt() is " + Math.sqrt(a));
}

public static void main(String[] args)
{
    new SquareRootRunner().run();
}
```
Newton's method (1)

- Find root of $f(x) = x^3 - x - 1 = 0$
- Since $f(1) = -1$ and $f(2) = 5$ there is a root between 1 and 2
  
  $x_0 = \text{guess}$

  $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- Calculate $x_0, x_1, x_2, \cdots$ until relative error between two approximations is small enough
public class NewtonRootCalculator1
{
    public void doIterations(double guess)
    {
        double xOld = guess;
        double xNew = guess - f(guess)/fPrime(guess);
        System.out.println(guess);
        System.out.println(xNew);
        while (Math.abs((xNew - xOld) / xNew) > 1E-16)
        {
            xOld = xNew;
            xNew = xOld - f(xOld) / fPrime(xOld);
            System.out.println(xNew);
        }
    }
}
// function whose root is to be found

private double f(double x)
{
    return x*x*x - x - 1.0;
}

// derivative function

private double fPrime(double x)
{
    return 3.0*x*x - 1.0;
}

Results

1.5
1.3478260869565217
1.325200398950907
1.3247181739990537
1.3247179572447898
1.324717957244746
1.324717957244746

initial guess

best approximation to the root
Double your money

Problem: How many months does it take to double an initial investment of \( x \) dollars if the annual interest rate is \( r\% \), and interest is compounded monthly?

Let initial amount be \( V \)

At the end of a month the new value of \( V \) is \( V + rV \)

\[ = V(1+r) \]

\( r\% \) annually corresponds to \( r/100/12 \) monthly

There is a formula for the answer but let's pretend we don't know it.
Double your money algorithm

ALGORITHM doublingTime\( (x, r) \)

\( \text{monthlyRate} \leftarrow r / 1200 \)
\( \text{month} \leftarrow 0 \)
\( \text{value} \leftarrow x \)

WHILE value < 2x DO

\( \text{month} \leftarrow \text{month} + 1 \)
\( \text{value} \leftarrow \text{value} \times (1 + \text{monthlyRate}) \)

END WHILE

RETURN \( \text{month} \)
public int doublingTime(double initialValue, 
    double annualRate) 
{ 
    double value = initialValue; 
    double monthlyRate = annualRate / 100.0 / 12.0; 
    int month = 0; 
    while (value < 2.0 * initialValue) 
    { 
        month = month + 1; 
        value = value * (1.0 + monthlyRate); 
    } 
    return month; 
}
public class DoubleYourMoney
{
    public int doublingTime(double initialValue,
                             double annualRate)
    {
        
        
    }
}
public class DoubleYourMoneyRunner
{
    public void run()
    {
        Scanner input = new Scanner(System.in);
        DoubleYourMoney calculator =
            new DoubleYourMoney();
        System.out.println("Enter initial amount");
        double amount = input.nextDouble();
        input.nextLine(); // eat end of line
        System.out.println("Enter annual rate in %");
        double rate = input.nextDouble();
        input.nextLine(); // eat end of line
        int totalMonths =
            calculator.doublingTime(amount, rate);
        // display results here
    }
    public static void main(String[] args)
    {
        new DoubleYourMoneyRunner().run();
    }
}
Factorization of an integer

- Every integer $n > 1$ can be expressed as a product of prime numbers:
  \[ n = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k} \text{ where } p_1 < p_2 < \cdots < p_k \]

- Examples:
  \[ 140931360 = 2^5 \times 3^3 \times 5 \times 17 \times 19 \times 101 \]
  \[ 140931369 = 3^2 \times 239 \times 65519 \]
Sketch of algorithm

- Given an integer n
  - Let \( q = n \) be the quotient initially
  - Initialize a list to empty
  - If \( t = 2 \) is a factor add it to the list and divide it out of \( q \) to get a new quotient \( q \)
  - If \( t \) is not a factor try the next one \( t = 3 \)
  - Continue and stop when \( t > \sqrt{q} \) since the largest possible factor \( t \) satisfies \( t \leq \sqrt{q} \)
- In Java we can use a string as a list
Factorization algorithm

ALGORITHM factor(n)

q ← n  // initialize quotient

// initialize trial factor

t ← 2

WHILE  t ≤ \( \sqrt{q} \) DO

IF t divides q THEN

Save t as a factor of q

q ← q \text{ div } t

ELSE

\( t ← \text{next trial factor} \)

END IF

END WHILE

Save q as last factor

---

we don't change t here since it may be a factor more than once

This will be 3 if t is 2 otherwise it will be \( t + 2 \) since all trial factors beginning with 3 will be odd
public String factor(int n) {
    int q = n; // initial quotient
    int t = 2; // initial trial factor
    String factors = "<"; // store factors in string
    while (t <= q / t) // don't need square root
    {
        if (q % t == 0) // t is a factor
        {
            factors = factors + t + ",";
            q = q / t;
        }
        else
        {
            t = (t == 2) ? 3 : t + 2;
        }
    }
    factors = factors + q + ">";
    return factors;
}
public class Factorizer
{
    public String factor(int n)
    {
        ...
    }
}

put factor method here
import java.util.Scanner;
public class FactorizerRunner
{
    public void displayFactors(int n)
    {
        Factorizer f = new Factorizer();
        for (int k = n; k <= n + 9; k++)
        {
            String factors = f.factor(k);
            System.out.println(k + " = " + factors);
        }
    }

    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter first value of n");
        int n = input.nextInt();
        input.nextInt(); // eat end of line
        input.nextLine(); // eat end of line
        new FactorizerRunner().displayFactors(n);
    }
}
java FactorizerRunner
Enter first value of n
140931360
140931360 = <2,2,2,2,2,3,3,3,5,17,19,101>
140931361 = <227,383,1621>
140931362 = <2,11,11,13,44797>
140931363 = <3,31,1515391>
140931364 = <2,2,7,157,32059>
140931365 = <5,571,49363>
140931366 = <2,3,23488561>
140931367 = <353,399239>
140931368 = <2,2,2,17616421>
140931369 = <3,3,239,65519>
140931370 = <2,5,3533,3989>
Sentinel-controlled while loop

This means that a special value called a sentinel value is used to terminate a while loop.

When the sentinel is found in the input data the loop is terminated.

Get an input value
WHILE input value is not the sentinel value DO
  statements
  Get another input value
END WHILE
Example

Read a sequence of marks in the range 0 to 100, calculate their average and stop if a negative mark is entered

Initialize sum and number of marks
read a mark
WHILE mark >= 0 DO
    IF mark <= 100 THEN
        update sum and number of marks
    ELSE
        Report error for marks > 100
    END IF
    Read another mark
END WHILE
Compute the average
public void averageMark()
{
    Scanner input = new Scanner(System.in);
    double sum = 0.0;
    int numberOfMarks = 0;
    double mark;

    System.out.println("Enter mark (neg to quit) ");
    mark = input.nextDouble();
    input.nextLine(); // eat end of line
    while (mark >= 0.0)
    {
        if (mark <= 100.0)
        {
            sum = sum + mark;
            numberOfMarks = numberOfMarks + 1;
        }
    }
else
{
    System.out.println("Marks > 100 invalid");
}
System.out.println("Enter mark (neg to quit)");  
mark = input.nextDouble();
input.nextLine();  // eat end of line
}  // end while

System.out.println("Averate mark is " +
    sum / numberOfMarks);
}  // end method
```java
import java.util.Scanner;
public class AverageMarkCalculator {
    public void averageMark() {
        ...
    }

    public static void main(String[] args) {
        new AverageMarkCalculator().averageMark();
    }
}
```

put averageMark method here
Query-controlled while loop

- If there is not sentinel value in an interactive while loop then it is necessary to ask the user before each input if the loop should be continued or terminated.

```java
public boolean moreValues()
{
    System.out.println("Continue (Y/N) ?");
    String reply = input.nextLine();
    return reply.equals("") ||
           reply.toUpperCase().charAt(0) == "Y";
}
```

```java
while (moreValues())
{
    // read a value and process it
}
```
public BankAccount findMaxBalance()
{
    BankAccount maxAccount = readAccount();
    while (moreAccounts())
    {
        BankAccount next = readAccount();
        if (next.getBalance() > maxAccount.getBalance())
        {
            maxAccount = next;
        }
    }
    return maxAccount;
}
maxAccount

next

BankAccount

balance

1000

BankAccount

balance

2000
import java.util.Scanner;
public class MaxBalanceCalculator {
    Scanner input = new Scanner(System.in);

    public BankAccount findMaxBalance() {...}
    private boolean moreAccounts() {...}
    private BankAccount readAccount() {...}

    public static void main(String[] args) {
        MaxBalanceCalculator calc =
            new MaxBalanceCalculator();
        System.out.println("Account with max balance is "+ calc.findMaxBalance());
    }
}
Pseudo-code do-while loop

REPEAT

Statements

WHILE BooleanExpression

REPEAT

Statements

UNTIL BooleanExpression
The do while loop in Java

- Not as common as the while loop
- The loop always executes at least once and the test for termination is made at the end of the statements in the loop:

```java
do {
    statements
} while (booleanExpression);
```

semi-colon is needed here
Flow chart for do-while

- **Expression**
  - true:
    - **Statements**
    - statements are executed at least once
  - false:
Count up to 10 with do-while

```java
int count = 1;
do {
    System.out.println(count + " ");
    count = count + 1; //or count++;
} while (count <= 10);
```

- The output is 1 2 3 4 5 6 7 8 9 10
Count down to 0 with do-while

```java
int count = 10;
do {
    System.out.println(count + " ");
    count = count - 1; // or count--;
} while (count > 0);
```

- Output is 10 9 8 7 6 5 4 3 9 2 1
General loop structure (1)

- while: exit at the top
- do while: exit at the bottom
- Following loop can exit anywhere

```
LOOP
  Statements A
  IF BooleanExpression THEN EXIT
  Statements B
END LOOP
```
General loop structure (2)

- Java version using the `break` statement which cause the loop to terminate.

```java
while (true)
{
    Statements A
    if (BooleanExpression) break;
    Statements B
}
```
Pseudo-code for-loop (up)

```
FOR k ← start TO end BY step DO
  Statements
END FOR
```

Assuming that \( step > 0 \) this loop counts up in steps of size given by \( step \) and ending with the last value of \( k \) such that \( k \leq end \). The successive values of \( k \) are \( start, start + step, start + 2*step, ... \), If \( start > end \) the for-loop is ignored.

If \( step \) has the value 1, then BY \( step \) can be omitted.
Pseudo-code for-loop (down)

FOR $k \leftarrow start$ TO $end$ BY $-step$ DO
    Statements
END FOR

Assuming that $step > 0$ this loop counts down in steps of size given by $step$ and ending with the last value of $k$ such that $k \geq end$. The successive values of $k$ are $start, start - step, start - 2*step, \ldots$, if $start < end$ the for-loop is ignored.
The Java for-loop

for ( Initialization ; Test ; Update )
{
Statements
}

Unfortunately the Java for-loop comes from C and is completely unreadable unless you restrict yourself to the specific pseudo-code cases on the next slides.
Counting up with for-loop

```
FOR k ← start TO end BY step DO
    Statements
END FOR
```

```
for (int k = start; k <= end; k += step) {
    Statements
}
```

For steps of 1 use `k++` instead of `k += step`
Counting down with for-loop

FOR \( k \leftarrow \text{start} \) TO \( \text{end} \) BY \(-\text{step}\) DO

Statements

END FOR

for (int k = start; k >= end; k -= step) {
    Statements
}

For steps of -1 use \( k-- \) instead of \( k -= \text{step} \)
counting up/down 1 to 10

For (int k = 1; k <= 10; k++)
{
    System.out.print(k + " ");
}

Count up from 1 to 10

For (int k = 10; k >= 1; k--)
{
    System.out.print(k + " ");
}

Count down from 10 to 1
counting down by 3

```java
for (int k = 10; k > 0; k -= 3) {
    System.out.print(k + " ");
}
```

The values displayed are 10 7 4 1
for (int k = 0; k <= 9; k++)
{
    double y = 10.0 + 20.0*k;
    g2D.draw(new Line2D.Double(10,y,200,y));
}
Computing $1 + 2 + 3 + \ldots + n$

Assuming $n$ is a positive integer the statements

```java
int sum = 0; // initialize sum
for (int k = 1; k <= n; k++)
{
    sum = sum + k;
}
```

compute the sum $1 + 2 + \ldots + n$
Birthday problem (1)

In a group of \( n \) people, what is the probability that two have the same birthday?

\[
p = 1 - \left( \frac{364}{365} \right) \left( \frac{363}{365} \right) \cdots \left( \frac{365 - k}{365} \right) \cdots \left( \frac{365 - n + 1}{365} \right)
\]

```java
public class BirthdayCalculator {
    public double probability(int n) {
        double product = 1.0;
        for (int k = 1; k <= n-1; k++)
            product = product * (365.0 - k)/365.0;
        return 1.0 - product;
    }
}
```
Include a method to display a table for \( n = \text{min} \) to \( n = \text{max} \).

```java
public void makeTable(int min, int max)
{
    for (int n = min, n <= max; n++)
    {
        System.out.println("p(" + n + ") = "
                       + probability(n));
    }
}
```
Computing factorials

\[
n! = \begin{cases} 
1 \times 2 \times \cdots n & \text{if } n > 0 \\
1 & \text{if } n = 0 
\end{cases}
\]

We can write a java method that uses a for loop to compute \( n! \) by initializing a product variable to 1 and multiplying it successively by 2, 3, ..., \( n \) in the loop.
A factorial method

int factorial(int n)
{
    int product = 1;
    for (int k = 2; k <= n; k++)
    {
        product = product * k;
    }
    return product;
}

overflow occurs if n > 12

try it with BeanShell

This works for n=0 and n = 1 since the for-loop is never executed in these cases.
public class FactorialCalculator
{
    public int factorial(int n)
    {
        if ( n < 0 || n > 12)
        {
            throw new IllegalArgumentException("...");
        }
        int product = 1;
        for (int k = 2; k <= n; k++)
        {
            product = product * k;
        }
        return product;
    }
}
import java.util.Scanner;
public class FactorialRunner {
    public void run() {
        Scanner input = new Scanner(System.in);
        FactorialCalculator calc =
                new FactorialCalculator();
        System.out.println("Enter value of n");
        int n = input.nextInt();
        System.out.println(n + "! = " +
                calc.factorial(n));
    }

    public static void main(String[] args) {
        new FactorialRunner().run();
    }
}
BigInteger class

- This class is in the java.math package so its full name is `java.math.BigInteger`.
- It provides methods for working with arbitrarily large integer called "big integers".
BigInteger Class summary (1)

public static BigInteger
valueOf(long val)

- make a big integer object from a long value

Example:
BigInteger bigI =
    BigInteger.valueOf(1);
converts the number 1 to big integer.
BigInteger Class summary (2)

- public BigInteger(String val)
  
  Constructor to make a big integer from a numeric string
  
  Example:
  BigInteger bigI = new BigInteger("111111111111111111111");
BigInteger Class summary (3)

- public BigInteger multiply(BigInteger val)
  - Multiply this big integer by another one to get a big integer. Example
  - BigInteger b1 =
    new BigInteger("1111111111111");
  - BigInteger b2 =
    new BigInteger("2222222222222");
  - BigInteger b3 = b1.multiply(b2);
  - Similarly for add, subtract, divide
BigInteger Class summary (4)

- public String toString()
  - Converts a big integer to a string
  - it can be used to display a big integer
public BigInteger(int n)
{
    BigInteger product = BigInteger.valueOf(1);
    for (int k = 2; k <= n; k++)
    {
        BigInteger bigK = BigInteger.valueOf(k);
        product = product.multiply(bigK);
    }
    return product;
}
BigInteger display method

display s in lines of width given by width

```java
public void displayLongString(String s, int width)
{
    int length = s.length();
    int numberOfLines = length / width;

    for (int k = 0; k < numberOfLines; k++)
    {
        System.out.println(s.substring(
            k * width, (k+1) * width));
    }
    if (length % width != 0) // partial line at end
        System.out.println(s.substring(
            numberOfLines * width));
}
```
import java.math.BigInteger;
public class BigFactorialCalculator {
    public void displayFactorial(int n) {
        String s = bigFactorial(n).toString();
        System.out.println("Number of digits is "+
                           s.length());
        System.out.println(n + "! = ");
        displayLongString(s, 60);
    }
    private BigInteger bigFactorial(int n) { ... }
    private void displayLongString(String s, int width) { ... }
}
import java.util.Scanner;
public class BigFactorialRunner {
    public void run() {
        Scanner input = new Scanner(System.in);
        BigFactorialCalculator calc = new BigFactorialCalculator();
        System.out.println("Enter value of n");
        int n = input.nextInt();
        calc.displayFactorial(n);
    }
    public static void main(String[] args) {
        new BigFactorialRunner().run();
    }
}
Example

```java
java BigFactorialRunner
Enter value of n
200
200! =
788 ........
... 0000
```

Try it in project
c:/book-projects/chapter7/factorial
Expressing for-loop as while

```plaintext
for ( Initialization ; Test ; Update )
{
    Statements
}
```

```plaintext
while ( Test )
{
    Statements
    Update
}
```
Loan repayment problem

Given the amount of a loan (principal), the number of years to pay back the loan, the number of payments per year, and the annual interest rate, produce a loan repayment table for each payment period showing the principal repaid and the principal remaining.
Financial mathematics

\[ A \quad \text{amount of the loan} \]
\[ y \quad \text{number of years to pay back loan} \]
\[ m \quad \text{number payments per year (periods per year)} \]
\[ j \quad \text{annual interest rate as a decimal number} \]
\[ n = my \quad \text{total number of payments} \]
\[ i = \frac{j}{m} \quad \text{interest rate per period as decimal number} \]
\[ R = \frac{A}{a(n,i)} \quad \text{payment made at end of each period} \]

\[ a(n,i) = \frac{1-(1+i)^{-n}}{i} \]
Algorithm

**ALGORITHM** LoanRepayment(loanAmount, years, paymentsPerYear, annualRate)

\[ n = paymentsPerYear \times years \]

\[ i = annualRate / paymentsPerYear \]

\[ payment = loanAmount / a(n,i) \]

\[ principalRemaining = loanAmount \]

**FOR** paymentNumber \( \leftarrow 1 \) **TO** \( n \) **DO**

\[ interest = principalRemaining \times i \]

\[ principalRepaid = payment - interest \]

\[ principalRemaining = principalRemaining - principalRepaid \]

**OUTPUT** paymentNumber, payment, interest, principalRepaid, principalRemaining

**END FOR**
$10,000 load, 10% per year, period of 5 years, payments twice a year.

<table>
<thead>
<tr>
<th>Payment Number</th>
<th>Payment</th>
<th>Interest Paid</th>
<th>Principal Repaid</th>
<th>Principal Remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1295.05</td>
<td>500.00</td>
<td>795.05</td>
<td>9204.95</td>
</tr>
<tr>
<td>2</td>
<td>1295.05</td>
<td>460.25</td>
<td>834.80</td>
<td>8370.16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>1295.05</td>
<td>120.40</td>
<td>1174.64</td>
<td>1233.38</td>
</tr>
<tr>
<td>10</td>
<td>1295.05</td>
<td>61.67</td>
<td>1233.38</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Generate the table as one large string with newlines in it to indicate the end of each line.
The StringBuilder class

- Use it instead of string concatenation for efficiency if you are doing a lot of string calculations.

- `public StringBuilder(int size)`
  - construct empty buffer for size characters

- `public void append(String s)`
  - append string s to end of buffer

- `public String toString()`
  - convert buffer to a String object when done
public class LoanRepaymentTable
{
    private double loanAmount; // initial amount
    private int year; // to pay back loan
    private int paymentsPerYear;
    private double annualRate; // as a fraction
    private String table; // repayment table
    public LoanRepaymentTable(double a, int y, int p, double r) {...}
    public String toString() {...}
    // private helper methods go here
}
LoanRepaymentTable (2)

- See LoanRepaymentTable class in BlueJ project:
  - book-projects/chapter7/loan-repayment

In Java 5 the String.format method can be used to do all the formatting of the table.
import java.util.Scanner;
public class LoanRepaymentTableRunner {
    public void run() {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter loan amount");
        double a = input.nextDouble();
        input.nextLine();
        System.out.println("Enter number of years");
        int y = input.nextInt();
        input.nextLine();
        System.out.println("Enter payments per year");
        int p = input.nextInt();
        input.nextLine();
        System.out.println("Enter rate in percent");
        double r = input.readDouble();
        input.nextLine();
        LoanRepaymentTable table =
            new LoanRepaymentTable(a, y, p, r);
        System.out.println(table);
    }
}
public static void main(String[] args) {
    new LoanRepayentTableRunner().run();
}

so we can run the class outside the Bluej environment
Nested loops

- Having one nested loop inside another is very common when processing two dimensional data such as a table
- Nested for loops are the most common form of nested loops
5 rows of 10 circles

double size = 40.0;
for (int row = 0; row <= 4; row++)
{
    for (int col = 0; col <= 9; col++)
    {
        double xTopLeft = size * col;
        double yTopLeft = size * row;
        g2D.draw(new Ellipse2D.Double(xTopLeft,
                                      yTopLeft, size, size));
    }
}

better to do this outside inner loop
A 4 by 10 square pattern

for (int row = 1; row <= 4; row++)
{
    for (int col = 1; col <= 10; col++)
    {
        System.out.print("* ");
    }
    System.out.println();
}

output is
4 rows of 10 asterisks
A triangular pattern

```java
for (int row = 1; row <= 4; row++)
{
    for (int col = 1; col <= row; col++)
    {
        System.out.print("*");  
    }
    System.out.println();
}
```

Note how the inner loop index depends on the outer loop index.
Computing powers (1)

- Write loops to compute the following table

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th></th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>16</td>
<td>25</td>
<td>100</td>
<td>100000</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>27</td>
<td>64</td>
<td>125</td>
<td>1000</td>
<td>1000000</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>16</td>
<td>81</td>
<td>256</td>
<td>625</td>
<td>10000</td>
<td>10000000</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>25</td>
<td>243</td>
<td>1024</td>
<td>3125</td>
<td>100000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>100</td>
<td>3125</td>
<td>1024</td>
<td>625</td>
<td>100000</td>
<td></td>
</tr>
</tbody>
</table>

Each row gives the powers of the number in the first column.
Computing powers (2)

```java
int val;
for (int n = 1; n <= 10; n++)
{
    System.out.printf("%5d",n);
    val = n;
    for (int p = 2; p <= 5; p++)
    {
        val = val * n;
        System.out.printf("%8d", val);
    }
    System.out.println();
}
```
Investment table problem

- Given an initial investment amount, compute a future value table for different rates from a minimum of rate of 4% to a maximum rate of 10% in steps of 0.5%, and for an investment time of 5 to 30 years in steps of 5 years. The rows of the table correspond to the rates and the columns correspond to the number of years.
**Investment table format**

<table>
<thead>
<tr>
<th>RATE</th>
<th>5 YEARS</th>
<th>10 YEARS</th>
<th>15 YEARS</th>
<th>...</th>
<th>30 YEARS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.00</td>
<td>1221.00</td>
<td>1490.83</td>
<td>1820.30</td>
<td>...</td>
<td>3313.50</td>
</tr>
<tr>
<td>4.50</td>
<td>1251.00</td>
<td>1566.99</td>
<td>1961.56</td>
<td>...</td>
<td>3847.70</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9.50</td>
<td>1605.01</td>
<td>2576.06</td>
<td>4134.59</td>
<td>17094.86</td>
<td></td>
</tr>
<tr>
<td>10.00</td>
<td>1645.31</td>
<td>2707.04</td>
<td>4453.92</td>
<td>19837.40</td>
<td></td>
</tr>
</tbody>
</table>

double small = 0.00001;
for (double rate = minRate; rate <= maxRate + small; rate *= rateStep)
{
    ...
        for (int years = minYears; years <= maxYears; years += yearStep)
            {
                ...
            }

}
InvestmentTable class

- InvestmentTable class
- InvestmentTableRunner class
- See BlueJ project
  - /book-projects/chapter7/investment
import java.util.Scanner;
public class InvestmentTableRunner
{
    public void run()
    {
        Scanner input = new Scanner(System.in);
        // read the input data here
        InvestmentTable table = new InvestmentTable(
                minRate, maxRate, rateStep, minYears,
                maxYears, yearStep, amount);
        System.out.println(table);
    }
    public static void main(String[] args)
    {
        new InvestmentTableRunner().run();
    }
}
Drawing Graph of a function

\[ x_0 = x_L, x_1 = x_L + dx, \ldots, x_i = x_L + i \, dx, \ldots, \]
\[ x_R = x_L + n \, dx \]

approximate curve by a sequence of line segments
Pseudo-code Algorithm

ALGORITHM DrawGraph($x_L, x_R, n$)

dx ← ($x_R - x_L$) / $n$

$(x_0, y_0) ← (x_L, y_L)$

FOR $i ← 1$ TO $n$ DO

$(x, y) ← (x_L + i \cdot dx, f(x_L + i \cdot dx))$

Draw line from $(x_0, y_0)$ to $(x, y)$

$(x_0, y_0) ← (x, y)$

END FOR
Sine curve

- Draw the graph of $\sin x$ from $x_L = -2\pi$ to $x_R = 2\pi$
Transformation

User space

Device space (pixels)

$$(x_{\text{max}}, y_{\text{max}})$$

$$(0,0)$$

$$(x_{\text{min}}, y_{\text{min}})$$

$$(w-1, h-1)$$
import library.GraphicsFrame;
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class SineGraph extends JPanel {
    private double xLeft = -2 * Math.PI;
    private xRight xRight = 2 * Math.PI;
    private double yBottom = -1.0;
    private double yTop = 1.0;
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2D = (Graphics2D) g;

    int w = getWidth();
    int h = getHeight();
    int numPoints = 100;

    double b = 1.1;
    AffineTransform world = worldTransform(xLeft, xRight, yBottom * b, yTop * b, w, h);
    g2D.transform(world);
g2D.setStroke(new BasicStroke(0.0f));
Line2D.Double xAxis =
    new Line2D.Double(xLeft, 0, xRight, 0);
Line2D.Double yAxis =
    new Line2D.Double(0,yBottom, 0, yTop);
g2D.setPaint(Color.blue);
g2D.draw(xAxis);
g2D.draw(yAxis);
double dx = (xRight - xLeft) / numPoints;
Point2D.Double p0 =
    new Point2D.Double(xLeft, Math.sin(xLeft));
g2D.setPaint(Color.black);
for (int i = 1; i <= numPoints; i++)
{
    double x = xLeft * i*dx;
    double y = Math.sin(x);
    Point2D.Double p1 = new Point2D.Double(x, y);
    g2.draw(new Line2D.Double(p0, p1));
    p0 = p1;
}
private AffineTransform worldTransform(
    double xMin, double xMax, double yMin,
    double yMax, int w, int h)
{  double sx = (w-1) / (xMax - xMin);
    double sy = (h-1) / (yMax - yMin);
    AffineTransform at = new AffineTransform();
    at.translate(0, h-1);
    at.scale(sx, -sy);
    at.translate(-xMin, -yMin);
    return at;
}
public void draw()
{  new GraphicsFrame("Graph of sin x",
    this, 301, 201);
}
public void main(String[] args) {...}
Recursive methods (1)

- Solve a problem in terms of a smaller version of itself

- Example (factorial function)
  
  \[0! = 0, \quad 1! = 1\] (base cases, \(n = 0, 1\))

  \[n! = n(n - 1)!\] (recursive cases, \(n > 1\))

- Example (Fibonacci numbers)

  \[F_0 = 0, \quad F_1 = 1\] (base cases, \(n = 0, 1\))

  \[F_n = F_{n-1} + F_{n-2}\] (recursive cases, \(n > 1\))
Recursive methods (1)

Example (gcd function)

\[ \text{gcd}(m, 0) = m, \quad \text{(base case, } n = 0) \]
\[ \text{gcd}(m, n) = \text{gcd}(n, m \mod n) \quad \text{(recursive cases, } n > 0) \]

Example \( \text{gcd}(2436, 1015) \)

\[ = \text{gcd}(1015, 2436 \mod 1015) = \text{gcd}(1015, 406) \]
\[ = \text{gcd}(406, 1015 \mod 406) = \text{gcd}(406, 203) \]
\[ = \text{gcd}(203, 406 \mod 203) = \text{gcd}(203, 0) \]
\[ = 203 \]
Recursive factorial method

```java
int factorial(int n)
{
    if (n == 0 || n == 1) // base case
        return 1;
    else // recursive case
        return n * factorial(n-1);
}
```

```java
public class FactorialCalculator
{
    public int factorial(int n) {...}
}
```

Can also test in using BeanShell workspace editor
import java.util.Scanner;
public class FactorialRunner {
    public void run() {
        Scanner input = new Scanner(System.in);
        FactorialCalculator calc = new FactorialCalculator();
        System.out.println("Enter value of n");
        int n = input.nextInt();
        System.out.println(n + "! = " + calc.factorial(n));
    }
    public static void main(String[] args) {
        new FactorialRunner().run();
    }
}
Recursive gcd method

```java
int gcd(int m, int n) {
    if (n == 0)
        return m;
    else
        return gcd(n, m % n);
}
```

```java
public class GcdCalculator {
    public int gcd(int m, int n) {...}
}
```

Can also test in using BeanShell workspace editor
import java.util.Scanner;
public class GcdRunner
{
    public void run()
    {
        Scanner input = new Scanner(System.in);
        GcdCalculator calc = new GcdCalculator();
        // read values of m, n here
        System.out.println("gcd(" + m + "," + n + ") = " + calc.gcd(m,n));
    }
    public static void main(String[] args)
    {
        new GcdRunner().run();
    }
}
Sum formulas

- Non-recursive sum method

\[ S(a, b) = a + (a + 1) + \cdots + b \]

- Recursive sum method

\[
\begin{align*}
S(a, a) &= a \quad \text{(base case } b = a) \\
S(a, b) &= a + S(a + 1, b) \quad \text{(recursive cases } b > a) 
\end{align*}
\]
Non-recursive sum method

```c
int sum(int a, int b)
{
    int s = 0;
    for (int k = a; k <= b; k++)
    {
        s = s + k;
    }
    return s;
}
```
Recursive sum method

int sum(int a, int b) {
    if (a == b) // base case
        return a;
    else
        return a + sum(a+1, b);
}