Before we begin

mkdir java
cd java
xemacs &

This very short introduction is only intending to give a quick start up procedure to use the basics of java. It absolutely does not constitutes a complete java course by itself. Much more information and tutorials can be found from SUN microsystem’s web site (http://java.sun.com).

1 Digital information

The information is encoded using two different status in electronical chips (electricity present or not). The unit of information is called a bit and it can represent 2 different values: 0 or 1. Usually, a computer does not manipulate individual bits (which carry very few information). Bits are gathered in bytes: packet of 8 bits. A byte can thus represent $2^8 = 256$ different values. Any piece of information in a computer is thus coded as a sequence of bits: 

...1 0 1 1 0 1 0 0 0 1 0 1 1 0 1 0 1 ...

Nothing differentiate two pieces of information. It is not possible to guess what it is supposed to encode. It may be numbers, text, image...anything that a computer can manipulate. We need types to describe the data content.
2 Types

2.1 Java basic types

There are mainly 5 basic types in java:

- **byte** (8 bits) it the fundamental information unit. It represents a numerical integer between 0 and 255 (256 possible values). A java byte is declared as:

  ```java
  byte b;
  ```

  b is a variable of type byte. It holds one value in the type interval. So far b as not been initialized so a default value is attributed by java (b=0). One can also declare and give a value to a variable at the same time:

  ```java
  byte v = 23;
  ```

  v is a byte which value is 23.

- Characters representing letters (in any alphabet), Chinese ideograms or so are encoded using 16 bits. For instance, the c character holding the letter capital A is defined as: `char c = 'A';` Note that the letter needs to be quoted.

- An integer is encoded using 32 bits. It is therefore bounded by the $2^{32}$ possible values. An integer can hold positive and negative values: `int i` declares an integer $i \in [-2^{31}, 2^{31} - 1]$.

- A real value is approximated by a 64 bits floating point number: `double d = 0.1; double e = 1e-6;`

- Text (best known as string of characters) is also represented by a specific kind of primitive type in java. The character string needs to be double quoted: `String s="this is text";`
2.2 Arrays

When more than a single value is used, arrays of values can be declared. Arrays can be created for any existing type using the \([]\) notation. Example:

```java
int[] array; // array is an (empty) array of integer
byte[] A = new byte[10]; // A is a 10 cells array of bytes
A[0] = 1; // one can assign values to any cell of A
...
A[9] = A[0]; // the value may be any byte
A[10] = 0; // however this line would cause an error
// at execution (best known as a java
// exception): there are only 10 cells
// into array A and there are numbered
// from 0 to 9. 10 falls out of the
// array’s bounds.
A[i] = j; // is valid as long as j is a byte and i
// belongs to [0, 9]
```

Note: indexes are numbered from 0 to length of the array minus 1.

2.3 Classes

When we need to manipulate pieces of information which are not primitive types of the language nor can be represented as arrays of the primitive types, we need to construct a new type which is a compound of the existing types. A java class defines a new type. For example, we want to manipulate complex numbers which are not part of the language. We can create a Complex class. Note that java requires the class Complex to be defined in a file with the same name and the java extension: Complex.java. Thus, create a file Complex.java, and let us define a complex as two real values: the real part and the imaginary part of the number.

```java
public class Complex {
    protected double real;
    protected double img;
}
```

real and img are two class variables better known as class attributes or class members. Note that we will declare all class members as protected for now.
3 Data processing

A class does not only defines types to hold values, it also contains methods (=function inside a class), used to process the class data. As an example, the method to square a real value is defined as:

```java
public double sqr (double d) {
    double r;
    r = d*d;
    return r;
}
```

A method is defined by its prototype and its body. In this example, the prototype is `double sqr(double d)`. It indicates:

- the method name `sqr`;
- the input values to be processed and their types: `(double d)`, a single real value designed as `d` in this case;
- and the type of the resulting value: the preceding `double` in this example indicates that the result of the processing will be a real value.

Note that we will declare all methods as `public` for now.

A method call is made by using the method name. For example:

```java
double v = sqr(3.2);
```

When this command is executed, the input parameter `d` of the method is assigned the value `3.2` indicated in the call. This value is squared `(d*d)` and stored in the temporary variable `r`. The result `r` is returned. Hence, the variable `v` which is assigned with the result receives the expected value `3.2^2`.

### 3.1 Constructor: a special method

A class defines a type (a range of possible values). An object is one particular value of this type. The constructor is a special method used to create and initialize objects of a given type. It is identified by the fact that:

- there is no return type;
- it has the same name as the class.
In the Complex class example, a very simple example is:

```java
public class Complex {
    protected double real;
    protected double img;

    public Complex () {
        real = 1.0;
        img = 0.0;
    }
}
```

This constructor has no input parameters. It always assigns the same value (1+0i) to the complex created. A new complex object is created when calling the `new` operator of the java language, followed by the constructor call:

```java
Complex c = new Complex(); // c value is 1 + 0i
```

A more useful constructor can assign any value (transmitted as input parameters) to the newly created object:

```java
public Complex(double r, double i) {
    real = r;
    img = i;
}
```

Thus the call:

```java
Complex c = new Complex(0.0, 1.0)
```

assigns the values 0.0 to parameter `r` and 1.0 to parameter `i`. These values are then copied into the class members `real` and `img`. The object value is then 0+1i as indicated by the caller.

### 3.2 The program entry point: the `main` special method

A computer program is an ordered sequence of operation that the microprocessor will strictly follow. A start point is needed. From the start point, all the instructions will then be executed in sequence. The entry point of any java program is the `main` method, added to one of the program classes, whose prototype is:
public static void main(String[] args) {
    To create a concrete program, let us add the following method to the Complex class which prints a complex value on the screen in the form \(x + yi\):

    public void print() {
        System.out.println(real + " + " + img + "i");
    }

    The `print` method makes a call to the `System.out.println` predefined system method that is used to print any string on the screen. Note that the "+" operator may be used to concatenate several strings.

    Here is a first complete program, including a `main` entry point:

    ```java
    public static void main(String[] args) {
        Complex c = new Complex();
        c.print();
        Complex c2 = new Complex(0.0, 1.0)
        c2.print();
    }
    ```

    It creates a first complex number `c`, prints its value \((1 + 0i)\), creates a second complex `c2` and prints its value \((0 + 1i)\).

    Note the doted notation to call a method on an object:

    ```java
    c.print();
    ```

    means calling the `print` method on the `c` object. `c` is prefixed in the call to indicate that it plays a particular role in the method call. This is the current object, on which the `print` method is called. Therefore, the value of `c` \((1 + 0i\) in this case) is to be printed. The same method can later be called on the `c2` object:

    ```java
    c2.print();
    ```

    a different value is printed, as `c2` has now become the current object.

    You can compile this program using the java compiler (javac):

    ```bash
    javac Complex.java
    ```

    And execute it for testing using the java interpreter (java):

    ```bash
    java Complex
    ```
3.3 More methods

Our complex class is currently limited to the creation of complex values and their printing on screen. To make it more useful, we want to add some complex arithmetic methods. Java does not know anything about complex: it does not define arithmetic operators on such values by itself. We will have to define the meaning of the operators through new methods.

The addition method for instance can be defined as:

```java
public void add(Complex other) {
    real = real + other.real;
    img = img + other.img;
}
```

This method is taking on input parameter (other). It may be called as:

c.add(c2);

In this call, the c2 parameter is added to the c current object. c2’s value is transmitted to the other temporary variable. The real part of the current complex (real) is assigned with the sum of its value and the other complex real part (other.real). Note that the doted notation is used for accessing a class member as well as a class method.

Also note that this method is defined such that it modifies the value of the current pixel: after calling the method, c’s value equals the sum of c’s initial value and c2 value. The original value is lost in this process. This is an arbitrary choice and the method could have been defined so that neither c nor c2 values are modified. In this case, a new object needs to be created to hold the resulting value:

```java
public Complex add2(Complex other) {
    Complex result = new Complex();
    result.real = real + other.real;
    result.img = img + other.img;
    return result;
}
```

The substraction method is very similar to the addition:

```java
public void substract(Complex other) {
    real = real - other.real;
}
```
img = img - other.img;
}

The multiplication method is also close:

```java
public void mult(Complex other) {
    double r = real * other.real - img * other.img;
    img = real * other.img + img * other.real;
    real = r;
}
```

Yet, be careful of defining the temporary value r to ensure that the real part of the current complex is not modified by the first line computation before being reused in the second line for the computation of the imaginary part. The naive definition below:

```java
public void mult(Complex other) {
    real = real * other.real - img*other.img;
    img = real * other.img + img*other.real;
}
```

would not work. Try for example with \((1 + 2i) \times (2 + 3i) = -4 + 7i\). The real value of the current complex is originally 1. The false method assigns the value \(-4\) to the real member of the current complex and then reuses this value (instead of 1) for computing the imaginary part. The computed imaginary part is then \(-8i\)!

## 4 Type conversion

Numerical values cannot always be converted safely from one type to another. For instance the following will cause an error:

```java
double d = 5.2;
int i = d;
```

because the real value d cannot be stored in the integer value i. If you want to enforce a conversion, you can use the (type) parenthesized operator. for instance:

```java
int i = (int) d;
```

will work. It will truncate the d real value (it does not correspond to a rounding but a truncation of the decimal part).
5 Control structures

To go further and implement more complex algorithms, control structures are needed that can modify the sequential flow of the program. The main control structures are tests (if and switch) that are used to take decision depending on a value and loops (for and while) which enable repetitive actions.

5.1 if... else... tests

An if test looks like:

```java
if (condition) {
    action if true...
}
else {
    action if false...
}
```

where condition is an arithmetic test, action if true is the sequence of commands to execute if the test happens to be positive, and action if false is the sequence of commands to execute if the test happens to be negative.

For example in our complex class, the print method can be modified using an if test to improve the printing of negative imaginary values:

```java
public void print() {
    if(img >= 0) {
        System.out.println(real + " + " + img + "i");
    }
    else {
        System.out.println(real + " - " + (-img) + "i");
    }
}
```

5.2 Multiple cases switch

The following control structure:
int i = 1;
switch(i) {
    case 0:
        ....
        break;
    case 1:
        ....
        break;
    case 2:
        ....
        break;
}

is a kind of extended if where multiple actions can be taken depending on multiple values of an input integer (i in this case). If the value of i is n, then the case n will be executed. When the execution flow reaches the break instruction, it then resumes just after the switch statement termination.

5.3 Loops

A for loop example is:

for(int i = 0; i < 10; i++) {
    System.out.println(i);
}

It reads as "for i = 0, while i < 10, incrementing i at each step". The body of the for (a simple print out of a value in this case), is iterated 0, 1 or more times. The number of iterations is defined by the 3 parameters of the for:

int i = 0 initialize i to zero
i < 10 iterate while i is lesser than 10
i++ increment i at each iteration

At each iteration, the value of i does indeed change and can be used as such in the loop.

A while loop is a close structure:
int i = 0;
while(i < 10) {
    System.out.println(i);
    i++;
}

In this case, you need to increment i in the loop.

Note that a for or a while loop may never terminate if the loop test never becomes true (e.g. when omitting to increment i, its value is constantly zero and therefore always lesser than 10).

6 The Java library

Many methods and classes are already defined in the very rich java library. In fact we have already been using one class of this library. The class String, which represents some text, is defined as an array of character. We have also already been using the method:

void System.out.println(String text)

to display text on the screen.

Among the available classes, the Math class defines many useful functions:

class Math {
    public final double PI;  // pi constant
    public double sqrt(double v);  // real square route
    public double exp(double v);  // real exponential
    public double pow(double x, double y);  // real power
    public double sin(double a);  // real sine
    ... cos, tan, acos, asin, atan ...
    public double atan2(double y, double x)
        // returns theta such as x = cos(theta) and y = sin(theta)
}

Methods may be called by prefixing them with the name of the class:

double d = Math.sqrt(2.0);
Using the java library, let us add a power method to the Complex class. The naive method would be to use a loop over the multiplication function. However, it would only be able to compute integer powers and it would prove to be very unefficient for large values of the power $n$. We will build a more elaborated method using the property:

$$(x + iy)^n = (r.e^{i\theta})^n = r^n.e^{in\theta}$$

The method thus needs:

1. to compute the modulus and the argument of the complex
2. to raise the modulus at the power $n$ and to multiply the argument by $n$
3. to find the Cartesian coordinates corresponding to this new complex

It stands as:

```java
public void power(double n) {
    double r = Math.sqrt(real*real + img*img);
    double t = Math.atan2(img, real);
    r = Math.pow(r, n);
    t = n * t;
    real = r * Math.cos(t);
    img = r * Math.sin(t);
}
```