2.2. Methods and control structures

Nassi - Shneiderman diagram

a) Sequential steps

Osnovni element NS dyagrama je jedan STEP (korak) i predstavljen je pravougaonikom.

x = 3.7
y = 5.9
z = 7.5 * x - y
Output z

```
double x, y, z;
x = 3.7;
y = 5.9;
z = 7.5 * x - y;
System.out.println("Value of z: " + z);
```

↳ Iz NS dijagrama se izrazi nekada mogu direktno prebaciti u Java-u.

Variable se deklarišu samo u Javi, ne i u NS dyagramu

↳ Jedna posebna stavka je statement Output, koji u Java code-u predstavlja „print statement"

Each algorithm takes one or more values as input and computes a result which is returned to the user of the algorithm. This is illustrated by a very simple algorithm which just adds two integers.

add2(a: int, b: int): int
Return a + b

```
public int add2(int a, int b) {
  return a + b;
}
```

ime_algoritma (lista inputa): tip_rezultata

x: double

↓

ime: tip

return,

takozvani specijalan izraz u strukturnom dijagramu koji indicira da se sljedeća vrijednost treba vratiti programu

↳ to se zove metoda

"Public" - bilo ko može koristiti metodu

označava tip izlaza

```
public int add2(int a, int b) {
    return a+b;
}
```

TIP IZLAZA

ova vrijednost se vraća programu

lista parametara (može biti i prazna)

```
add2(a:int, b:int):int
    return a+b;
```

```
add3(int a, int b, int c):int
    return a+b+c
```

MyMath class - sadrži matematičke funkcije, bez Main metode.

TestMyMath - za testiranje metoda

```
1 public static void main(String[] args) {
2    int r1;
3    MyMath mm = new MyMath();
4    r1 = mm.add2(3, 48);
5    System.out.println("mm.add2(3, 48) returns " + r1);
6 }
```

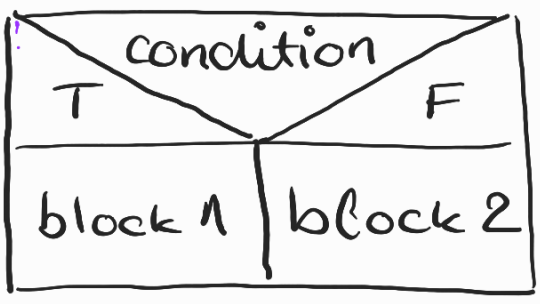
Metoda se poziva na sljedeći način

r1 = mm.add2(3,48)

- mm: object on which we apply the method
- add2: method we are invoking
- (3,48): ulazni parametri za metodu
- r1 =: return value is stored in this variable

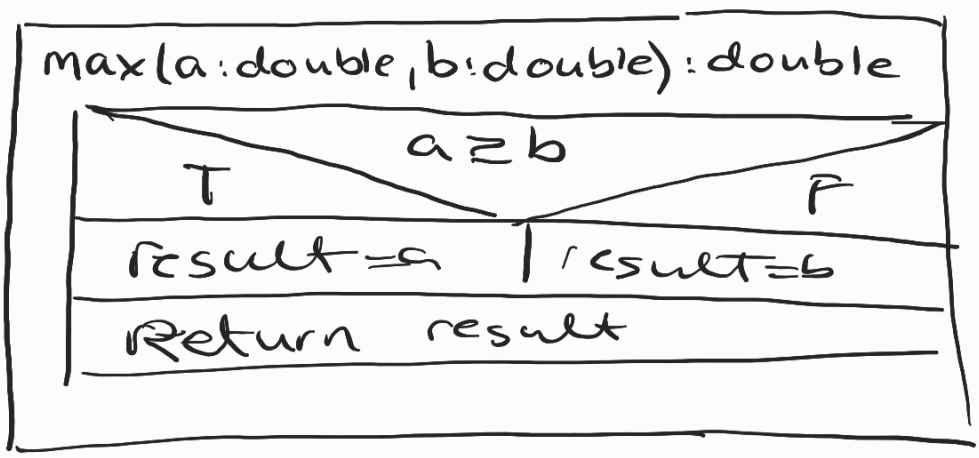
Selection

Uslov koji može biti true ili false (ispunjen ili ne) se koristi za odlučivanje da li se jedna sekvenca izraza izvršava ili druga




```
if (condition) {
    block 1 ;
} else {
    block 2 }
```

In Java, a conditional statement starts with the keyword `if` followed by the condition enclosed in parenthesis. The condition can be any expression evaluating to a boolean value. The block immediately after the condition is executed if the condition evaluates to `true`. After the first block you see the keyword `else` which is followed by the block to be executed otherwise. Note that the `else` clause is optional. The following example determines the maximum of two double values.




```
pubic double max(double a, double b) {
  double result ;
  if (a >= b) {
     result = a ;
  } else { result = b; }
  return result ;
    }
```

Postoji i metoda switch - nije ključna jer se ona to može izvršiti sa if i else

Indeterminate Loops

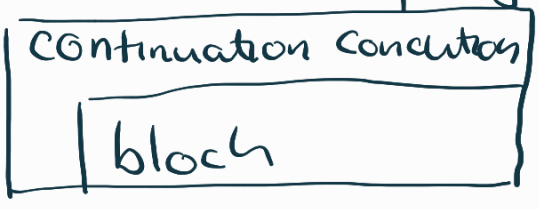
In Java, as in all programming languages, there are control structure that let you repeat statements. There are two forms for repeating loops that are best when you do not know how many times a loop should be processed (these are "indeterminate loops"). The `while` loop tests the continuation condition at the beginning whereas the `do/while` loop test the continuation condition at the end. Only the first form is introduced here.

Petlje - omogućavaju ponavljanje izraza više puta

"Indeterminate loops" → kada ne znamo koliko puta se neka petlja treba obaviti

"while" petlja - provjerava uslov na početku petlje

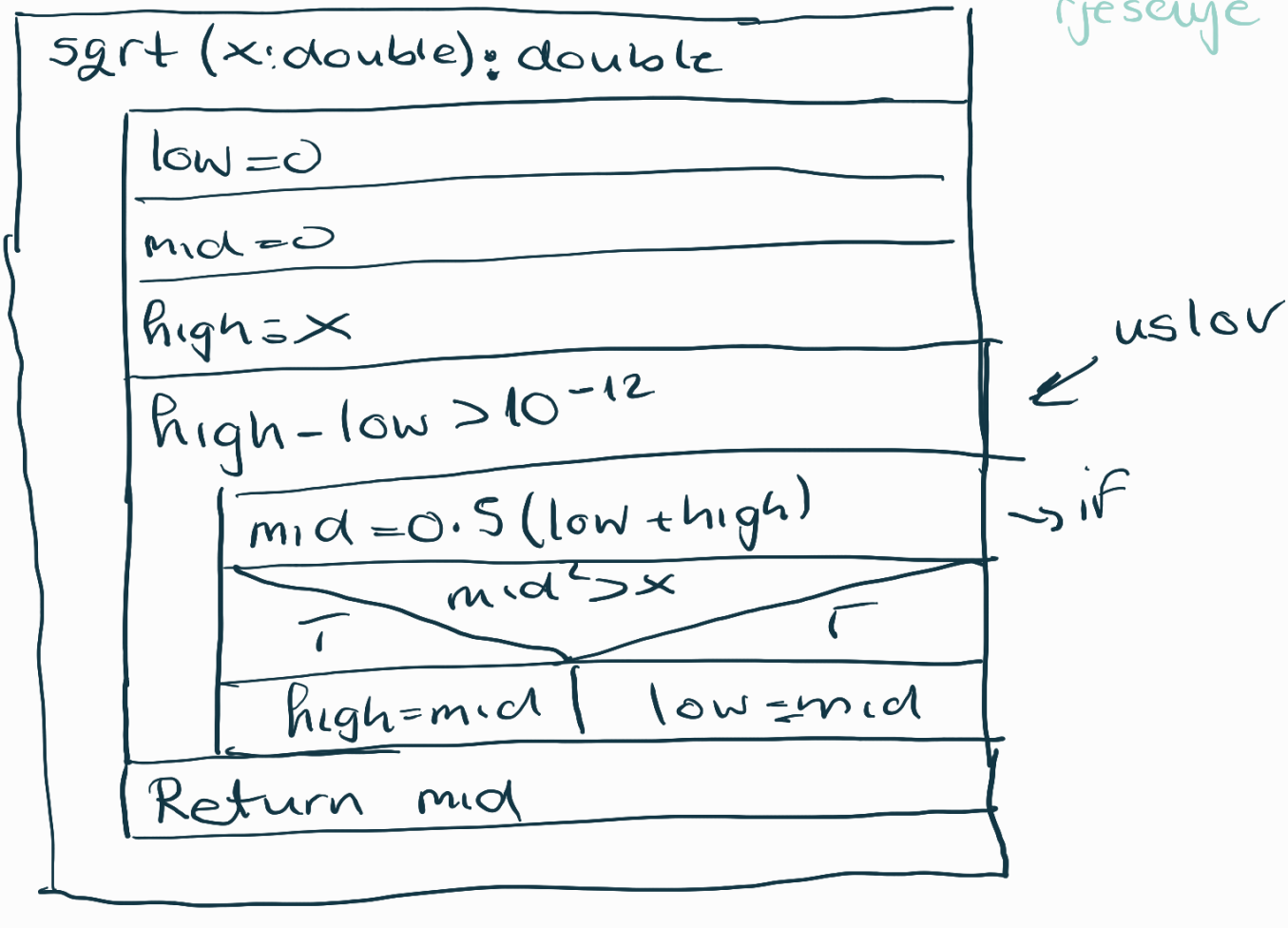
"do while" petlja - testira uslov na kraju petlje



while (continuation condition) {
block }

Često se primjenjuju za algoritme koji iterativno poboljšavaju približno rješenje

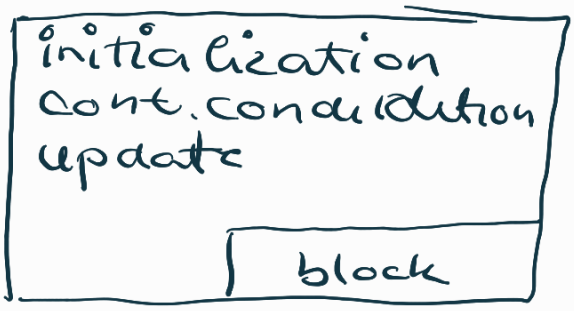
Primjer: "while" petlja



```
public double sqrt (double x) {
    double low = 0;
    double mid = 0;
    double high = x;
    while (high - low > 1e-12) {
        mid = 0.5 * (low + high);
        if (mid*mid > x) {
            high = mid; } else {
            low = mid; }
    }
    return mid;
}
```

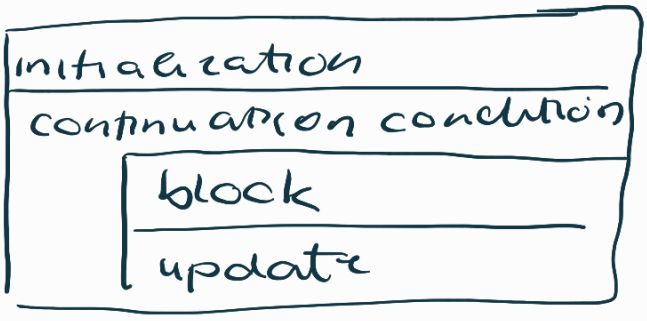
Determinate loops - koriste se kada je broj iteracija unaprijed poznat

The `for` loop is a very general construct to support iteration that is controlled by a counter which is updated after each iteration (*left* and *center*). Note that each `for` loop can be easily translated into an equivalent `while` loop (*right*). In Fig. 2.2, a `for` loop is employed to compute the factorial $n!$ of a nonnegative integer n.

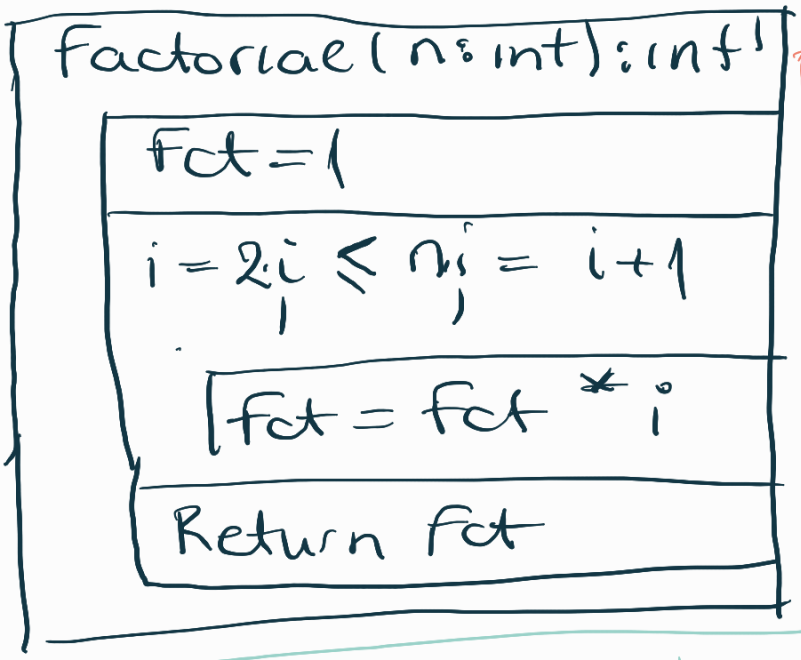



```
for (init.; cont.; update) {
    block; }
```

↳ svaka for petlja se može prevesti u „while" petlju



Primjer: Factorial




```
public int factorial(n=int) {
    int  fct=1
    For (int i=2; i=<n; i++){
      fct = fct * i;
    }
return fct; }
```

⟶ Trebamo koristit' double !!!

→ Factorial su veliki brojevi, tako da nam je potreban tip varyable koji može pohranit' što veće vrijednosti

2.3 Java Keywords

At the end of this section, a listing of all Java keywords is provided for completeness. The keywords you should be familiar with after this semester are typed in boldface.

abstract	default	**if**	**private**	**this**
boolean	**do**	**implements**	protected	**throw**
break	**double**	**import**	**public**	**throws**
byte	**else**	instanceof	**return**	transient
case	**extends**	**int**	short	**try**
catch	**final**	**interface**	**static**	**void**
char	finally	long	strictfp	volatile
class	float	native	**super**	**while**
const	**for**	**new**	switch	
continue	goto	package	synchronized	

Static methods

In order to understand static methods, let us quickly recall the way "normal" methods work. In a previous chapter, you created a `Box` object and changed its color:

```
1  Box box1 = new Box();
2  box1.setColor("red");
```

In the first line, the new `Box` object is constructed and stored in the variable `box1`. Then, in the second line you invoke the `setColor` method on your `Box` object that is stored in the variable `box1`. In short: "normal" methods operate on objects that you have constructed using `new`.

In contrast, static methods are methods that do not operate on objects. You do not need to create an object using `new` in order to invoke a static method. A good example for the use of static methods is the `Math` class. Because static methods belong to the class they are also called classifier methods. Normal methods, that operate on instances of a class are often called instance methods.

↳ "Normalne" metode djeluju na objekte koje smo konstruisali korištenjem "new".

↳ "Static methods" — ne djeluje na objekte, "ne moramo kreirati objekte" korištenjem "new" da bi pozvali statičke metode

↳ Dobar primjer je Math. classa, samo je pozvano npr. na broj bez kreiranja objekta

↳ Zato što pripadaju klasama, često ih zovemo "Classifier metodama"

↳ Normalne metode koje djeluju na instancama cela naziva se Instance metodama

When to use static methods? Sparingly. Static methods make sense for simple operations that only need a few parameters. Good examples can be found in the `Math` class. Also, static methods are used if only access to the static attributes of a class is needed.

↳ Koristimo za jednostavne operacije koje trebaju samo nekoliko parametara (npr. odredivanje apsolutne vrijednosti)
↳ Koriste se samo ako je potreban pristup statičkim atributima klase

2.3.2 Static attributes

If you define an attribute as `static`, that attribute only exists once. In contrast, "normal" attributes exist for each object that you construct. → normalni postoje za svaki

Statički atributi postoje samo jednom! objekat koji napravimo

Constants. If you qualify an attribute as `final`, the value of this attribute can not be changed after initialization. Often, static attributes are declared as `final` to define some predefined constant. An example is `Math.PI`. The definition of π in the `Math` class is

```
public static final double PI = 3.14159265358979323846;
```

You access a `public static` attribute by using the name of the containing class and the dot operator:

```
double u = 4 * Math.PI;
```

On the other hand, you get an error during compilation if you would write

```
Math.PI = 4.0;
```

This is because `PI` is qualified as `final` in the `Math` class. Finally, let us review the mysterious

```
System.out.println("Hello World");
```

`System` is a class that contains several useful static attributes and methods. One specific static attribute of the `System` class is `out`; it refers to an instance of the class `PrintStream` that represents a stream on which you can write. Have a look at the Javadoc to find out more about print streams.

↳ Npr. Ako je objekat trougao, onda će normalni atributi $x_1, x_2, x_3, y_1, y_2, y_3$ biti za svaki objekat tipa Triangle koji napravimo.
Statički bi bio za bilo koji parametar npr Math.abs(x) → za samo ovo x

↳ Finae → vrijednost atributa se ne može mijenjati nakon inicijalizacije. Ovako se često definišu pre-determinisane konstante poput PI (Math.PI)

↳ pozivaju se Math.PI

ime klase (Math) · dot operator (.)