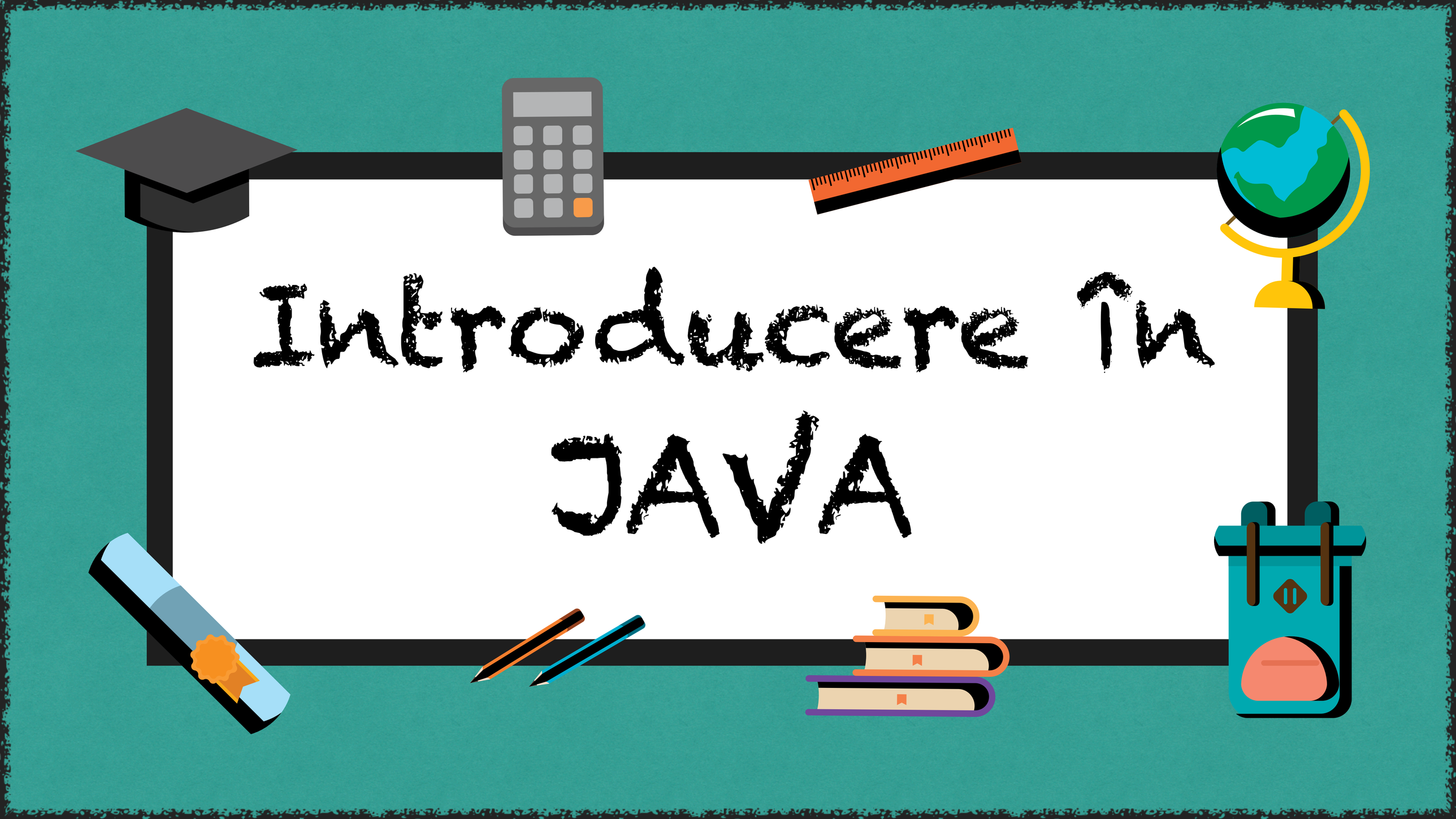

Introducere în
JAVA

Scurtu Diana

Găvănescu Cristina

Șchiopu Matei

Crăciun Roberta-Alexandra

Popa Alexandru-Cristian

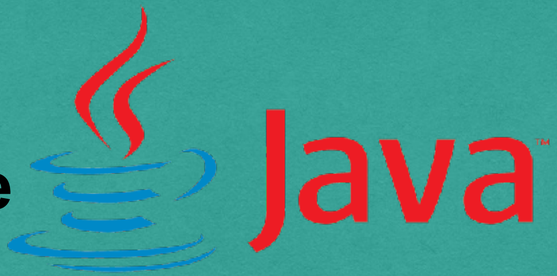
Ce este Java?

- o insulă din Indonezia

(126 650 km², 65 mil. locuitori)

- un jargon american pentru cafea

- o platformă şi un limbaj de programare orientat pe obiecte



În funcţie de modul de execuţie al programelor, limbajele de programare se împart în două categorii :

- **interpretate:** instrucţiunile sunt citite linie cu linie de un program numit interpretor şi traduse în instrucţiuni maşină; *avantaj:* simplitate; *dezavantaj:* viteza de execuţie redusă;
- **compilate:** codul sursă al programelor este transformat de compilator într-un cod ce poate fi executat direct de procesor; *avantaj:* execuţie rapidă; *dezavantaj:* lipsa portabilităţii, codul compilat într-un format de nivel scăzut nu poate fi rulat decât pe platforma pe care a fost compilat.

Programele Java sunt atât interpretate cât şi compilate

Codul de octeţi este diferit de codul maşină. Codul maşină este reprezentat de o succesiune de 0 şi 1; codurile de octeţi sunt seturi de instrucţiuni care seamănă cu codul scris în limbaj de asamblare. Codul maşină este executat direct de către procesor şi poate fi folosit numai pe platforma pe care a fost creat; codul de octeţi este interpretat de mediul Java şi de aceea poate fi rulat pe orice platformă care foloseşte mediul de execuţie Java.

Fazele prin care trece un program Java sunt:

```
Cod sursa Java -> (compilare) -> Cod de octeti -> (interpretare)
```

JDK vs JRE

JDK reprezintă **J**ava **D**eveloper **K**it, în timp ce **JRE** reprezintă mediul **J**ava **R**untime **E**nvironment.

JRE este ceea ce este necesar pentru **a rula aplicații Java**.

JDK include JRE și multe alte instrumente de dezvoltare a aplicațiilor Java. De exemplu, compilatorul Java (javac) este inclus numai în JDK.

Dacă doriți să rulați software Java, puteți pur și simplu descărca JRE.

Dacă sunteți **dezvoltatori** de software Java, trebuie să utilizați JDK.

C++	Java
C++ dispune de indicatori aritmetici (**pointers**).	Java **nu permite** crearea şi utilizarea de indicatori aritmetici (**pointers**).
Alocarea şi **delocare** memoriei este responsabilitatea programatorului.	De **alocarea** şi **delocarea** memoriei se ocupă **JVM** (Java Virtual Machine).
C++ **nu este un limbaj 100% orientat obiect**, astfel că poţi scrie un cod fără să foloseşti o clasă sau un obiect.	Java **este** prin definiţie **un limbaj de programare orientat obiect**. De aceea nu poţi scrie un program în Java fără să foloseşti **cel puţin o clasă**.
C++ are trei specificatori de acces: **privat**, **public** şi **protected**.	Java are patru specificatori de acces: **public**, **privat**, **protected** şi implicit (**default**).
C++ dispune de **constructori** şi de **destructori**.	Java are **numai constructori**. Destructorii nu sunt prevăzuţi în acest tip de limbaj.

Avantajele limbajului Java faţă de C++

- **Garbage Collection (GC)** = o formă automată de gestionare a memoriei. Acesta are tendinţa de a aduna surplusuri sau memorie ocupată de obiecte ce nu mai sunt necesare programului, contribuind la eliminarea grijii de alocare manuală a memoriei. Totodată contribuie la **diminuarea apariţiei erorilor de programare** (bugs) şi **îmbunătăţeşte timpul de execuţie**.

- **Procesul de construcţie** –> În comparaţie cu Java, construcţiile în C++ sunt mai complicate şi se mişcă mai lent. Programatorii spun că un produs integral elaborat folosind C++ poate fi realizat şi în **20 de ore**, în timp ce acelaşi produs construit cu Java poate fi gata şi în **mai puţin de 10 minute.**

- **Siguranţa** -> Java include şi o metodă de verificare a variabilelor care trebuie să rămână în anumite limite înainte de a fi folosite.

Există două posibilități de a lucra în Java:

- în linie de comandă
- folosind un editor Java cum ar fi:

NetBeans

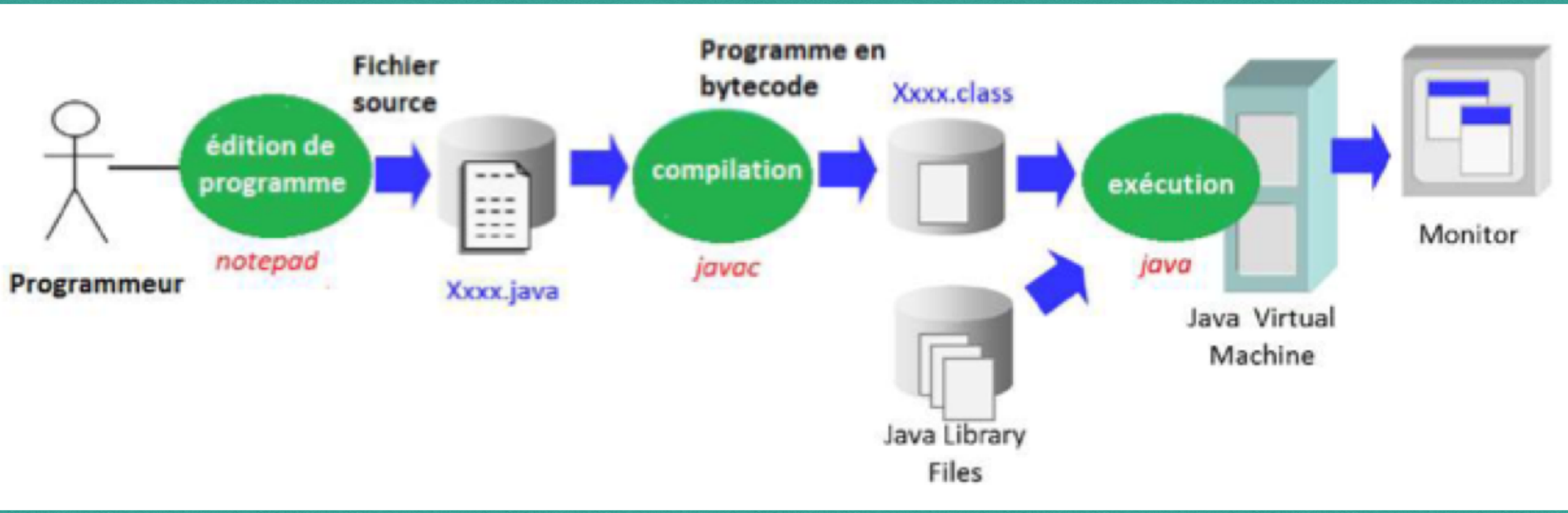
Linie de comandă

- Editarea programului într-un editor de texte;
- Salvarea programului sub numele `NumeClasa.java` unde *NumeClasa* este numele clasei care conţine metoda `main()`. Într-un program Java trebuie să existe o singură clasă care să conţină o metodă `main()`. Cu alte cuvinte, numele clasei trebuie să coincidă cu numele fişierului. Extensia fişierului este ***.java***
- Compilarea programului se face cu ajutorul comenzii

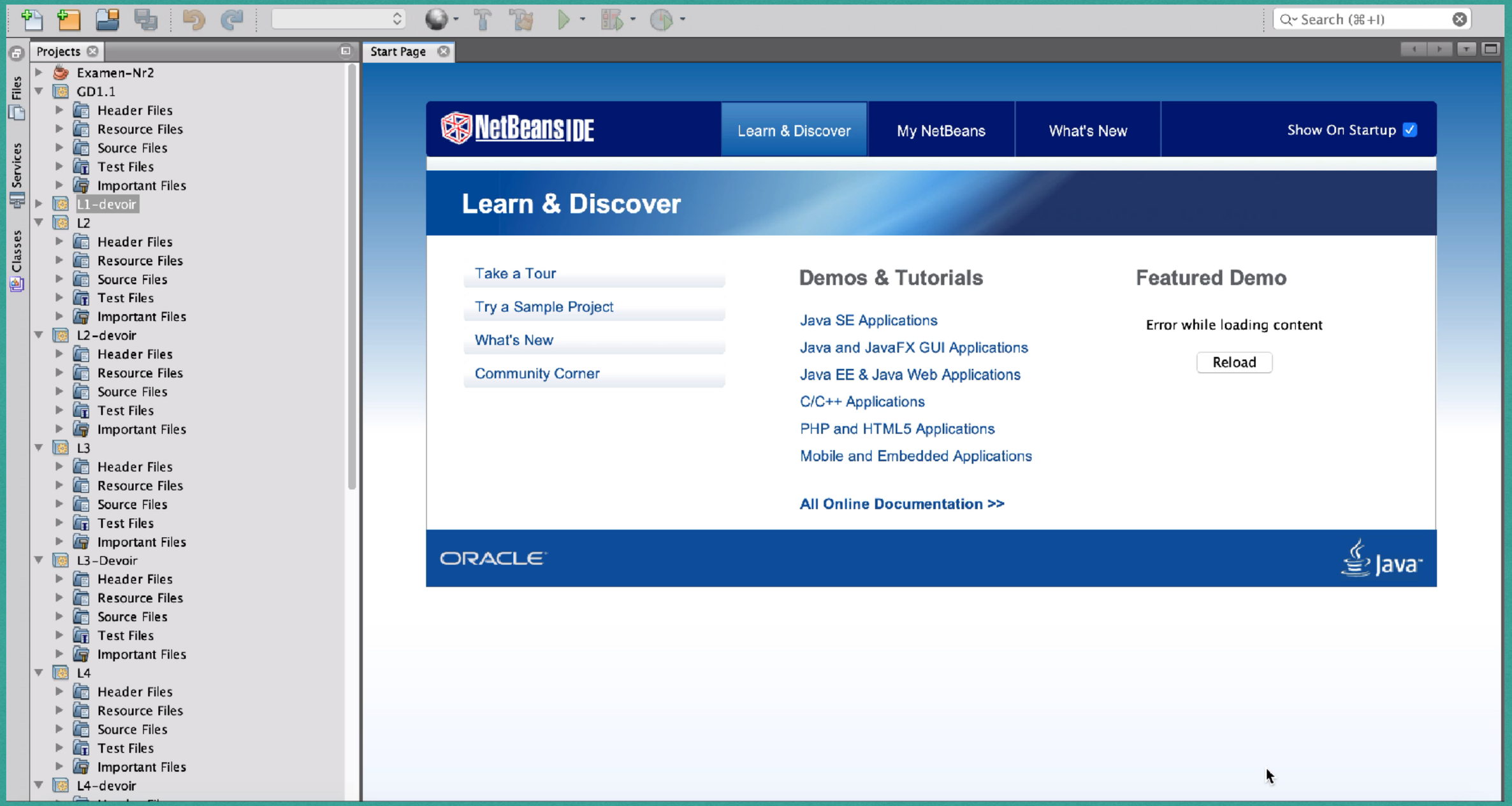
```
javac NumeClasa.java
```

- Executarea programului se face cu ajutorul comenzii

```
java NumeClasa
```


Programmeur
édition de programme
notepad
Fichier source
Xxxx.java
compilation
javac
Programme en bytecode
Xxxx.class
Java Library Files
exécution
java
Java Virtual Machine
Monitor

Primul program în Java



Operatori

= (semnul egal)

Operatorii de atribuire pot fi înlănţuiţi. De exemplu: `a=b=c=10`

+, -, *, /, %

Operatorii prescurtaţi sunt: **+=, -=, *=, /=, %=**

Exemplu: `n += 2` este echivalentă cu `n=n+2`

operatori aritmetici unari: +, ++ (incrementare), **--** (decrementare). Aceștia pot fi **prefixaţi (++x sau --x)** sau **postfixaţi (x++ sau x--).**

Exemple:

`-x` reprezintă opusul lui x

```
int x=5,y=7;
 x++; // x = 6
 y--; // y = 6
```

```
int x=5,y=7;
 ++x; // x = 6
 --y; // y = 6
```

```
int x=5,y;
 y=x++; // y = 5, x = 6
```

```
int x=5,y;
 y=x--; // y = 5, x = 4
```

```
int x=5,y;
 y=++x; // x = 6, y = 6
```

```
int x=5,y;
 y=--x; // x = 4, y = 4
```

&&(and), ||(or), !(not)

operatori relaţionali: <, <=, >, >=, ==, !=

operatori pe biţi: & (and), |(or), ^(xor), ~(not)

operatori de translaţie: <<, >>, >>> (shift-are la dreapta fără semn)

? :

```
expresie_logica ? expresie1 : expresie2
```

Valoarea expresiei este dată de `expresie1` dacă `expresie_logică` este true sau de `expresie2` dacă `expresie_logică` este false.

Exemplu: Metoda de calculul minimului a două numere este:

```
public int min (int a, int b){
  return a<b?a:b;
}
```

operatorul + pentru concatenarea şirurilor:

```
String s="abcd"
int x=100;
System.out.println(s + " - " + x); //afiseaza abcd-100
```

```
public class Operator{
  public static void main(String args[]){
   int a=2, b=3,c;
   c=a+b;
   System.out.println(a+" "+b+" “+c);
   a++; --b;
   System.out.println(a+" "+b+" "+c);
   c=++a+b++;
   System.out.println(a+" "+b+" "+c);
   a/=2;b*=2;
   System.out.println(a+" "+b+" "+c);
   c=a-- + --b;
   System.out.println(a+" "+b+" "+c);
   c=b=(a+=1);
   System.out.println(a+" "+b+" “+c);
  }
}
```

a	b	c
2	3	5
3	2	5
4	3	6
2	6	6
1	5	7
2	2	2

Să se calculeze maximul a două numere.

```
public class Maxim{
   public static void main(String args[]){

      int a=12, b=5,m;

      if (a>b) m=a;
      else m=b;

      System.out.println("Maximul dintre "+a+" si "+b+" este: "+m);
    }
}
```

Să se rezolve ecuaţia de gradul I ax+b=0 cunoscând coeficienţii a şi b.

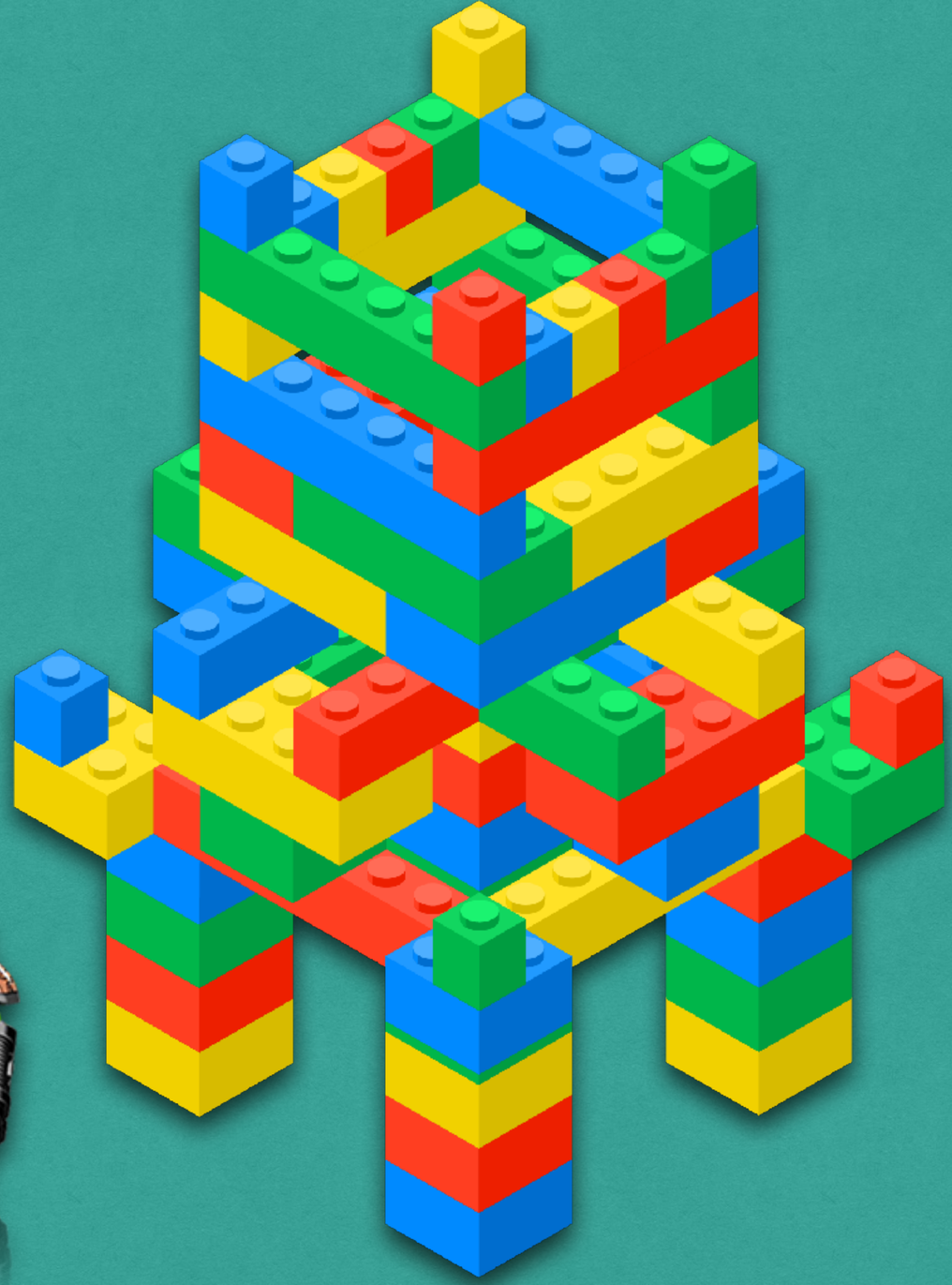
```
public class ecGrI{
   public static void main(String args[]){
       int a=12, b=5;
       System.out.print("Ecuatia: "+a+"x+"+b+"=0");
       if (a==0)
           if (b==0)
              System.out.println(" are o infinitate de
solutii");
          else
              System.out.println(" nu are solutii");
       else {
             double x =-(double)b/a;
             System.out.println(" are solutia "+x);
           }
     }
 }
```

Ecuația de gradul al II-lea ax^2+bx+c=0 cunoscând coeficienții a, b, c.

```
package ecuatiegr2;

public class EcuatieGr2 {

    public static void main(String[] args) {


        int A=1, B=-4, C=3;
        System.out.println("Numerele introduse sunt: "+A+", "+B+" si "+C);
        System.out.println("Solutile ecuatiei: ");
        if( A!=0 ){
            double delta = B*B-4*A*C;
            if( delta > 0 ){
                double x1=(-B+Math.sqrt(delta))/(2*A);
                double x2=(-B-Math.sqrt(delta))/(2*A);
                System.out.println("   x1="+x1);
                System.out.println("   x2="+x2);
            }
            else{
                double x = -B/(2*A);
                System.out.println("   x="+x);
                }
        }
        else System.out.println(" Nu avem ecuatie de gradul II");
    }
}
```

Clase și obiecte

Fiecare piesă de Lego este un mic **obiect** care împreună cu altele ajută la crearea altor **obiecte mai mari**. Exact aşa stau lucrurile şi în **programarea orientată pe obiecte(POO)**: obiecte mici puse împreună formează obiecte mari.



Clase vs obiecte

Când scriem programe într-un limbaj orientat pe obiecte, nu **definim** obiecte ci ***clase de obiecte***, unde o **clasă reprezintă un şablon pentru mai multe obiecte cu caracteristici similare**. Clasele întrupează toate caracteristicile unei mulţimi particulare de obiecte. **Culoarea** lego-ului, **mărimea** sa, indiferent dacă are umflături pe partea de sus sau lovituri de jos, dacă se poate conecta la alte lego-uri. Toate aceste atribute sunt statul lego.

Clase și obiecte

Apoi, lego-urile au, de asemenea, **comportament,** pe care mai târziu le vom numi **metode**.

Apoi, ce faceți cu toate aceste legi, creați alte metode care să spunem:

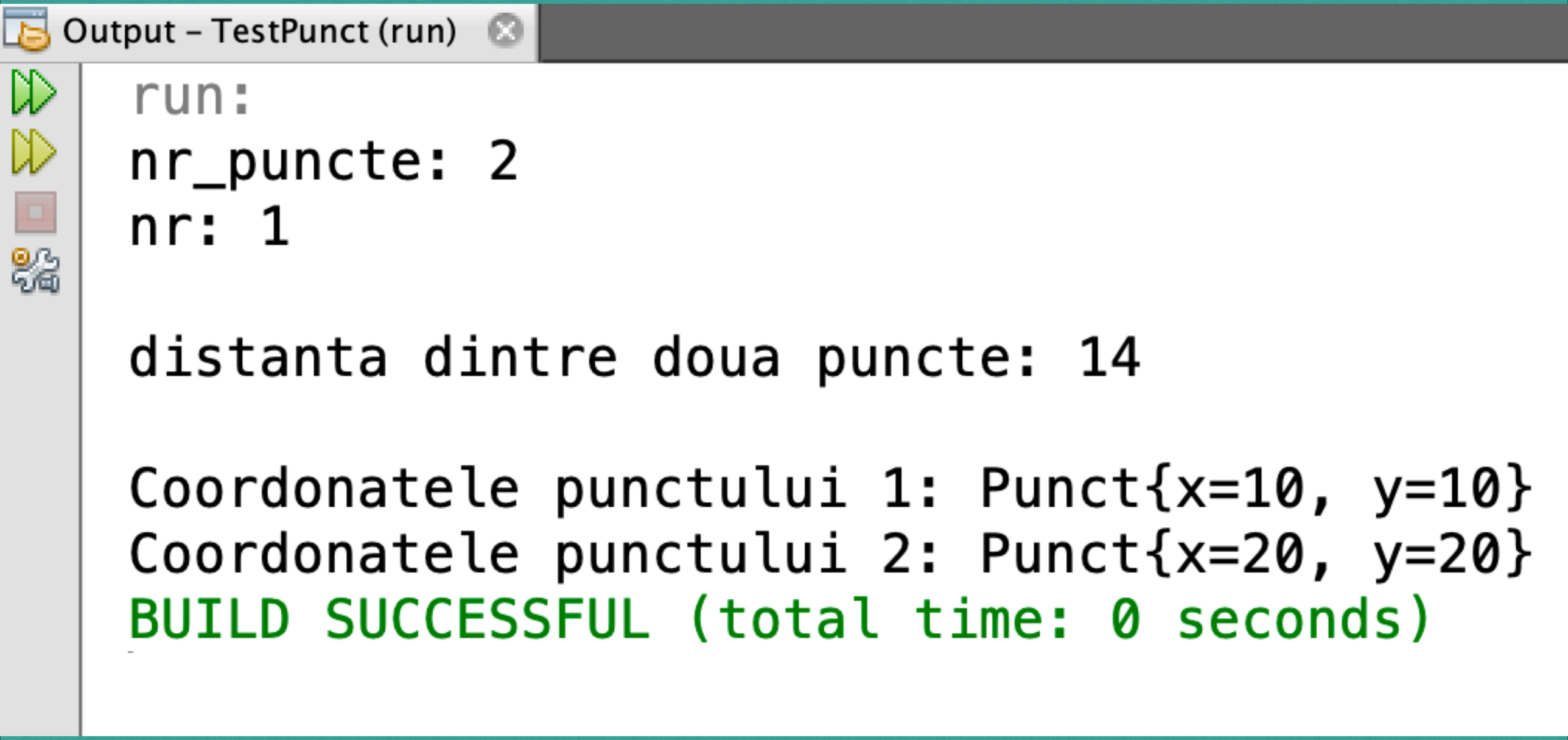
“adună toate aceste lego-uri pentru a construi o navă și un castel”

În cele din urmă, **metoda** finală este un fel de operațiune în care navele aruncă bile una contra celeilalte pentru a ajunge la castel.

```java
package testpunct;

class Punct{

    private int x, y;
    static int nr_puncte = 0;
    protected int nr = 0;

    // constructor vid
    public Punct(){}

    // constructor
    public Punct(int x, int y){
        this.x = x;
        this.y = y;

        nr_puncte++;
        nr++;
    }

    // metoda
    public int distanta_2pct(Punct a){

        return (int)Math.sqrt( Math.pow(this.x - a.x, 2) + Math.pow(this.y - a.y, 2) );
    }
```

```java
    // getter
    public int getNr(){ // cream un getter pt nr deoarece acesta este private si nu il putem obtine altfel in main
        return this.nr; // return nr;
    }

    // setter
    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }

    @Override
    public String toString() {
        return "Punct{" + "x=" + x + ", y=" + y + '}';
    }


}
```

```java
public class TestPunct {

    public static void main(String[] args) {

        Punct p1 = new Punct(10,10);
        Punct p2 = new Punct(20,20);

        System.out.println("nr_puncte: " + Punct.nr_puncte);
        System.out.println("nr: " + p2.getNr());
        System.out.println();
        System.out.println("distanta dintre doua puncte: " + p1.distanta_2pct(p2));
        System.out.println();
        System.out.println("Coordonatele punctului 1: " + p1.toString());
        System.out.println("Coordonatele punctului 2: " + p2.toString());
    }

}
```


Output – TestPunct (run)
run:
nr_puncte: 2
nr: 1
distanta dintre doua puncte: 14
Coordonatele punctului 1: Punct{x=10, y=10}
Coordonatele punctului 2: Punct{x=20, y=20}
BUILD SUCCESSFUL (total time: 0 seconds)