Install: Java + JDK + IDE (Apache NetBeans)

Variables i tipus de dades:

```
int edat = 30;
double altura = 1.75;
boolean esEstudiant = true;
char inicial = 'J';
String nom = "Maria";
```

Operadors:

```
int a = 10;
int b = 5;

int suma = a + b; // 15
int resta = a - b; // 5
int multiplicacio = a * b; // 50
int divisio = a / b; // 2
int modulo = a % b; // 0

boolean esMajor = a > b; // true
boolean esIgual = a == b; // false

int increment = a++; // increment = 10, a = 11
int decrement = b--; // decrement = 5, b = 4
```

Estructures de control - Condicions:

```
int edat = 18;

if (edat >= 18) {
   System.out.println("Ets major d'edat");
} else {
   System.out.println("Ets menor d'edat");
}
```

Estructures de control - Bucles:

```
// Bucle while
int contador = 0;
while (contador < 5) {
   System.out.println("Contador: " + contador);
   contador++;
}

// Bucle for
for (int i = 0; i < 5; i++) {
   System.out.println("i: " + i);
}
```

Classes i objectes:

```
// Definició d'una classe
class Persona {
    // Atributs
    String nom;
    int edat;

    // Mètode
    void saludar() {
        System.out.println("Hola, el meu nom és " + nom);
    }
}

// Creació d'un objecte
Persona persona1 = new Persona();
persona1.nom = "Joan";
persona1.edat = 25;
persona1.saludar(); // Imprimeix: Hola, el meu nom és Joan
```

La programació orientada a objectes (POO) és un paradigma de programació que organitza el codi al voltant d'objectes, els quals són instàncies de classes. A continuació, expliquem els conceptes clau de la POO:

1. **Objectes**: En la POO, un objecte és una instància d'una classe. Un objecte té propietats (anomenades atributs) i comportaments (anomenats mètodes). Per exemple, si tenim una classe "cotxe", un objecte d'aquesta classe podria tenir atributs com "marca", "model" i "color", i mètodes com "accelerar" i "frenar".

2. **Classes**: Les classes són les plantilles o els "plànols" per a crear objectes. Una classe defineix les propietats i els mètodes que tots els objectes de la mateixa classe tindran. Les classes permeten estandarditzar i reutilitzar codi, ja que es poden crear múltiples objectes a partir d'una mateixa classe. Utilitzant l'exemple anterior, la classe "cotxe" definiria les propietats i els mètodes comuns a tots els cotxes.

```
class Cotxe {
    // Atributs
    String marca;
    String model;
    String color;

    // Mètodes
    public void accelerar() {
        System.out.println("accelera cotxe");
    }
    public void frenar() {
        // Implementació del frenat
    }
}
```

3. **Herència**: És un concepte que permet crear noves classes a partir d'una classe existent, estenent o ampliant les seves característiques. La classe original es coneix com a classe mare o superclasse, mentre que les classes derivades es coneixen com a classes filles o subclasses. Amb l'herència, les subclasses hereten totes les propietats i els mètodes de la superclasse i poden afegir-hi nous atributs o mètodes o modificar els existents. Això permet establir jerarquies i organitzar millor el codi. Per exemple, podríem tenir una classe mare "vehicle" i subclasses com "cotxe", "moto" i "camió", que heretarien les característiques bàsiques de la classe mare.

```
class CotxeEsportiu extends Cotxe {
    // Atribut addicional
    int potencia;

    // Mètodes
    @Override
    public void accelerar() {
        System.out.println("accelera esportiu");
    }
    // Mètode addicional
    public void augmentarPotencia() {
        // Implementació de l'augment de potència
    }
}
```

4. **Polimorfisme**: És la capacitat de les classes filles d'implementar mètodes de la superclasse de manera diferent. Això significa que, tot i que tenen el mateix nom, els mètodes en subclasses poden tenir comportaments diferents. Això permet tractar objectes de diferents subclasses de manera genèrica utilitzant el tipus de la superclasse. Per exemple, si tenim una superclasse "animal" i subclasses com "gat", "gos" i "ocell", totes elles podrien tenir un mètode "ferSo" que produeix sons característics de cada animal.

```
public class Main {
   public static void main(String[] args) {
        Cotxe cotxeNormal = new Cotxe();
        CotxeEsportiu cotxeEsportiu = new CotxeEsportiu();

        // Tractar-los com a objectes de la classe pare
        Cotxe cotxe1 = cotxeNormal;
        Cotxe cotxe2 = cotxeEsportiu;

        cotxe1.accelerar(); // Crida al mètode de Cotxe
        cotxe2.accelerar(); // Crida al mètode de CotxeEsportiu
   }
}
```

En aquest exemple, tant "cotxeNormal" com "cotxeEsportiu" són tractats com a objectes de la classe pare "Cotxe". Però, gràcies al polimorfisme, la crida al mètode "accelerar()" es comporta diferent en funció del tipus d'objecte al que es refereix. Així, la crida "cotxe1.accelerar()" executa el mètode de la classe "Cotxe", mentre que "cotxe2.accelerar()" executa el mètode de la classe "CotxeEsportiu".