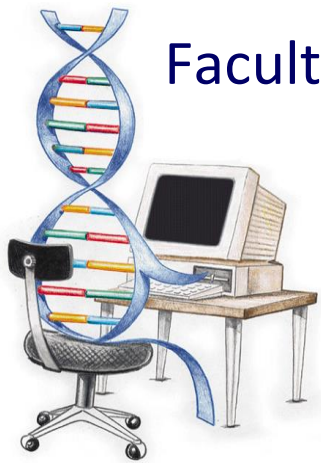


# ALGORITMOS EVOLUTIVOS

## Curso 2023

# Implementación de AEs usando jMetal

Centro de Cálculo, Instituto de Computación  
Facultad de Ingeniería, Universidad de la República, Uruguay



cecal



UNIVERSIDAD  
DE LA REPÚBLICA  
URUGUAY

## Características de jMetal

- jMetal proviene de Java METAheuristic ALgorithms
- Es una biblioteca orientada a objetos para abordar problemas de optimización con metaheurísticas
  - No es solamente para la implementación de AE
- Provee soporte para problemas multi-objetivo



### Referencias:

Durillo, J. J., & Nebro, A. J. (2011). jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10), 760-771.

Nebro, A. J., Durillo, J. J., & Vergne, M. (2015). Redesigning the jMetal multi-objective optimization framework. In *Proceedings of the companion publication of the 2015 annual conference on genetic and evolutionary computation* (pp. 1093-1100).

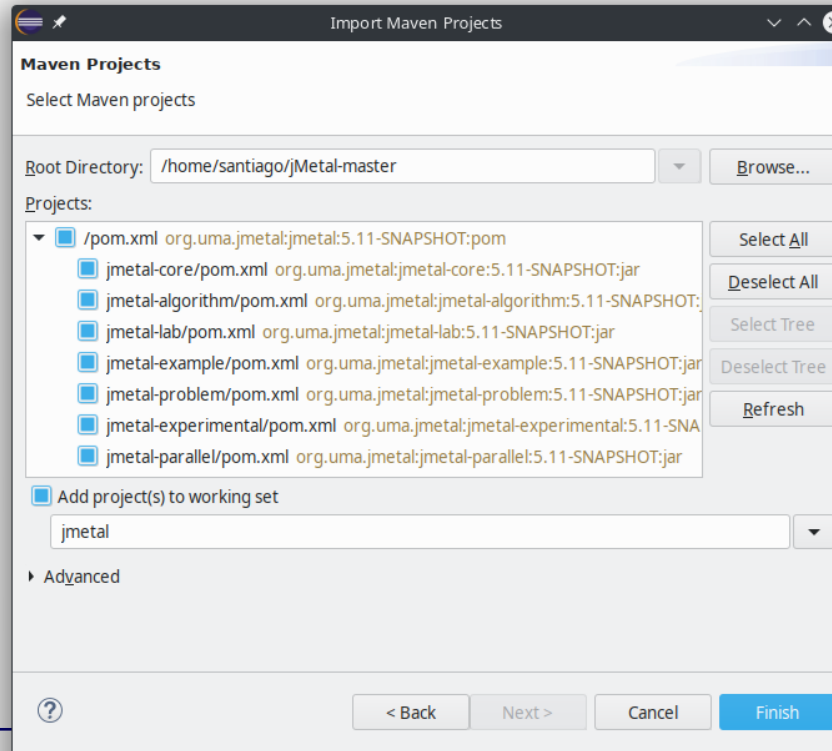
## Requerimientos

- Java 11 JDK (Java Development Kit) o mayor (<https://jdk.java.net/16/>)
- Maven es necesario para el manejo de dependencias (<https://maven.apache.org/>)
- Entorno de desarrollo (opcional pero *fuertemente* recomendado)
  - Eclipse (<https://www.eclipse.org/>)
  - *Otros*: NetBeans, IntelliJ IDEA, etc.



## Instalación

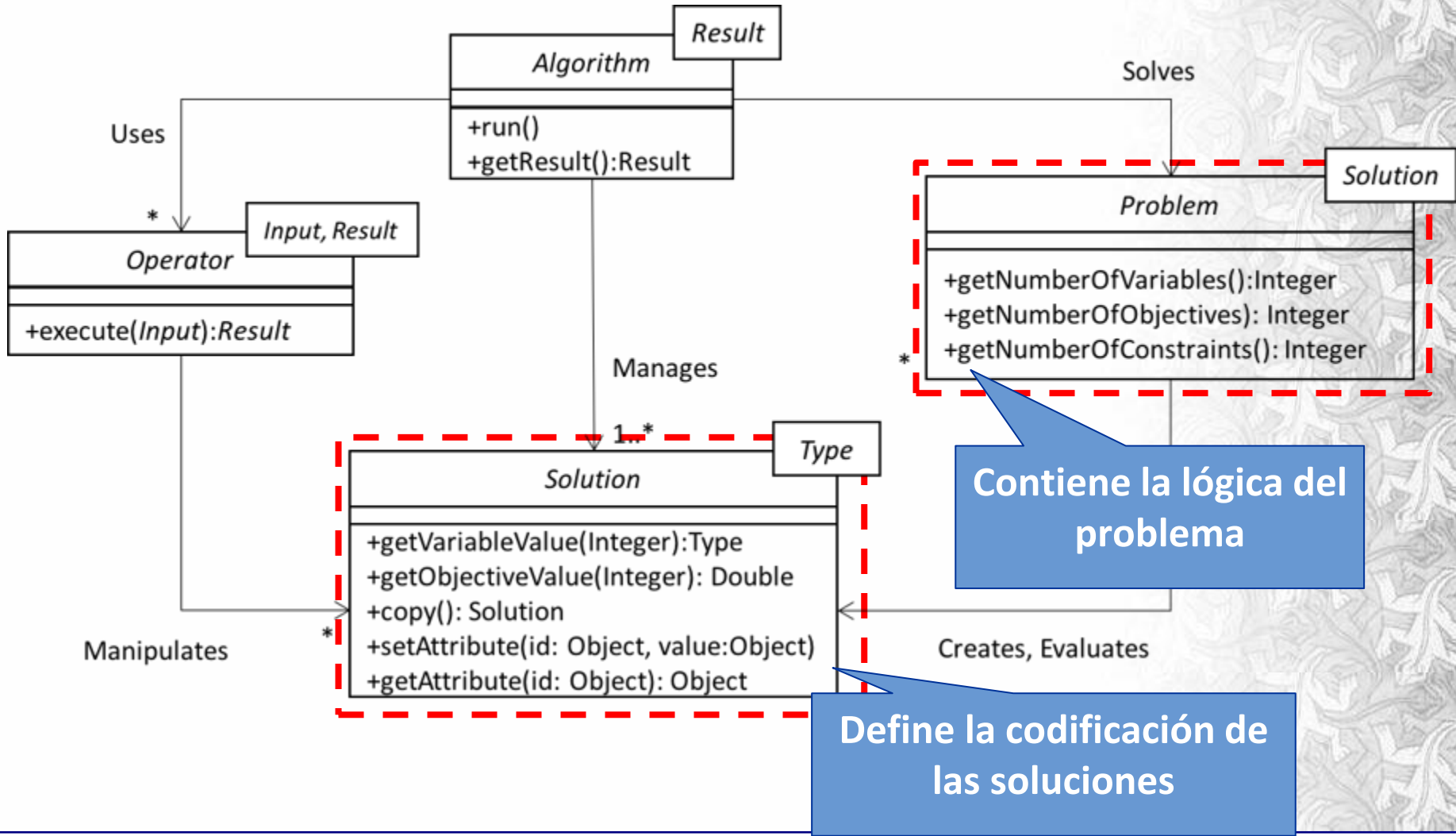
- Descargar y descomprimir la última versión (*estable*) de jMetal: <https://github.com/jMetal/jMetal/archive/refs/tags/jmetal-5.10.zip>
- Importar los proyectos a Eclipse con los siguientes pasos:  
**File > Import... > Maven > Existing Maven Projects**



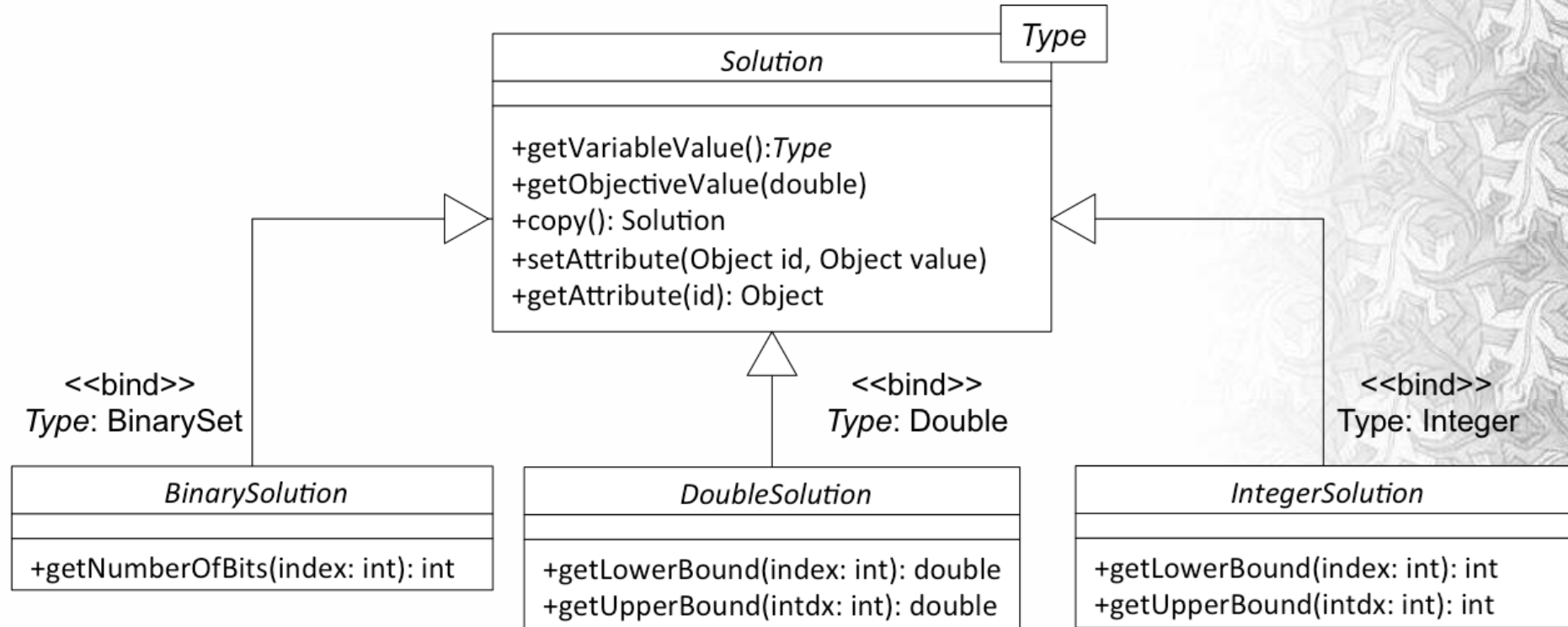


## Sub proyectos de jMetal

- **jmetal-core**: Clases principales de jMetal
  - Interfaces y clases abstractas, operadores, codificación de soluciones, indicadores de calidad, etc.
- **jmetal-algorithm**: Algoritmos disponibles
  - GA (generacional y de estado estacionario), NSGA-II, etc.
- **jmetal-problem**: Problemas de ejemplo y de benchmark
  - OneMax, StringMatching, etc.
- **jmetal-example**: Ejemplos de resolución de problemas
- **jmetal-lab**: Permite el diseño de estudios para comparar metaheurísticas
- **jmetal-experimental** y **jmetal-parallel**







Otros: **PermutationSolution**, **SequenceSolution**,  
**CompositeSolution**.

Algunos operadores disponibles:

- **Selección:** mejor; torneo binario y N-ario; aleatoria, etc.
- **Cruzamiento:** de uno, dos y N puntos; uniforme; PMX; SBX; etc.
- **Mutación:** cambio de bit; intercambio; polinomial; aleatoria simple; uniforme; etc.

Los operadores de mutación y cruzamiento están condicionados por el tipo de Solution



```
public interface Problem<S> extends Serializable {  
    /* Getters */  
    int getNumberOfVariables();  
    int getNumberOfObjectives();  
    int getNumberOfConstraints();  
    String getName();  
  
    /* Methods */  
    void evaluate(S solution);  
    S createSolution();  
}
```

Condicionado a la dirección de la optimización del algoritmo

Algunos modelos de problemas disponibles:

- **Binary problem:** `AbstractBinaryProblem`
- **Integer problem:** `AbstractIntegerProblem`
- **Double problem:** `AbstractDoubleProblem`
- **Permutation problem:** `AbstractIntegerPermutationProblem`

El objeto `Problem` es el indicado para contener la información de la instancia del problema a resolver

Algunos algoritmos disponibles:

- **Mono-objetivo:** Genetic algorithm, Particle swarm, Coral reef, Differential evolution, Evolution strategy.
- **Multi-objetivo:** NSGA-II, SPEA 2, ABYSS, IBEA, MOCell, MOCHC, MOEA/D, etc.

¡En general los algoritmos de jMetal resuelven problemas de minimización!

- **OneMax**
  - *Representación*: vector de bits
  - *Objetivo*: maximizar la cantidad de 1's
- **NIntegerMin**
  - *Representación*: vector de enteros
  - *Objetivo*: minimizar la sumatoria de la diferencia absoluta de cada entero de la representación con un valor de referencia
- Otros ejemplos
  - TSP, StringMatching, etc.

## Ejemplo: OneMax

```
public class OneMax extends AbstractBinaryProblem
{
    private int bits;

    public OneMax(Integer numberOfBits) {
        setNumberOfVariables(1);
        setNumberOfObjectives(1);
        setName("OneMax");
        bits = numberOfBits;
    }

    public BinarySolution createSolution() {
        return new DefaultBinarySolution(
            Arrays.asList(bits),
            getNumberOfObjectives());
    }
}
```

## Ejemplo: OneMax

```
public BinarySolution evaluate(BinarySolution solution)
{
    int counterOnes = 0;
    BitSet bitset = solution.variables().get(0);

    for (int i = 0; i < bitset.length(); i++) {
if (bitset.get(i)) {
        counterOnes++;
    }
    }

    // OneMax is a maximization problem:
    // multiply by -1 to minimize
    solution.objectives()[0] = -1.0 * counterOnes;
    return solution;
}
```





```
BinaryProblem problem = new OneMax(512);
```

```
SinglePointCrossover crossover;  
crossover = new SinglePointCrossover(0.9);
```

```
BitFlipMutation mutation;  
mutation = new  
    BitFlipMutation(1.0/problem.getBitsFromVariable(0));
```

```
BinaryTournamentSelection<BinarySolution> selection;  
selection = new  
    BinaryTournamentSelection<>();
```



## GenerationalGeneticAlgorithmBinaryEncodingRunner

```
Algorithm<BinarySolution> algorithm =  
  new GeneticAlgorithmBuilder<>(  
    problem, crossover, mutation)  
    .setPopulationSize(100)  
    .setMaxEvaluations(25000)  
    .setSelectionOperator(selection)  
    .build();
```

```
AlgorithmRunner algorithmRunner = new  
  AlgorithmRunner.Executor(algorithm).execute();
```

```
BinarySolution solution = algorithm.getResult();
```



## GenerationalGeneticAlgorithmBinaryEncodingRunner

```
List<BinarySolution> population = new ArrayList<>(1);  
population.add(solution);
```

```
new SolutionListOutput(population)  
    .setVarFileOutputContext(new  
        DefaultFileOutputContext("VAR.tsv"))  
    .setFunFileOutputContext(new  
        DefaultFileOutputContext("FUN.tsv"))  
    .print();
```

## Ejemplo: NIntegerMin

```
public class NIntegerMin extends AbstractIntegerProblem {  
    private int valueN;  
  
    public NIntegerMin(int numberOfVariables, int n,  
        int lowerBound, int upperBound) {  
        valueN = n ;  
        setNumberOfVariables(numberOfVariables);  
        setNumberOfObjectives(1);  
        setName("NIntegerMin");  
        List<Integer> lowerLimit =  
        new ArrayList<>(getNumberOfVariables());  
        List<Integer> upperLimit =  
        new ArrayList<>(getNumberOfVariables());  
  
        for (int i = 0; i < getNumberOfVariables(); i++) {  
            lowerLimit.add(lowerBound);  
            upperLimit.add(upperBound);  
        }  
    }  
}
```

## Ejemplo: NIntegerMin

```
public IntegerSolution evaluate(  
    IntegerSolution solution) {  
    int approximationToN = 0;  
  
    for (int i=0; i<solution.variables().size(); i++) {  
        int value = solution.variables().get(i);  
        approximationToN += Math.abs(valueN - value);  
    }  
  
    solution.objectives()[0] = approximationToN;  
    return solution ;  
}
```

- Manejo de restricciones dentro de la función **evaluate** en la clase **Problem**
- Algunas acciones que pueden tomarse son:
  - *Penalizar la solución*: modificar su campo **objectives**
  - *Corregir la solución*: modificar su campo **variables**
  - *“Descartar” la solución*: la solución debe ser reemplazada





## Links

### Java

<https://www.oracle.com/java/technologies/javase-jdk16-downloads.html>

### Eclipse

<https://www.eclipse.org/>

### Maven

<https://maven.apache.org/>

### jMetal

<https://github.com/jMetal/jMetal/tree/jmetal-5.10>