The University Of Sheffield.

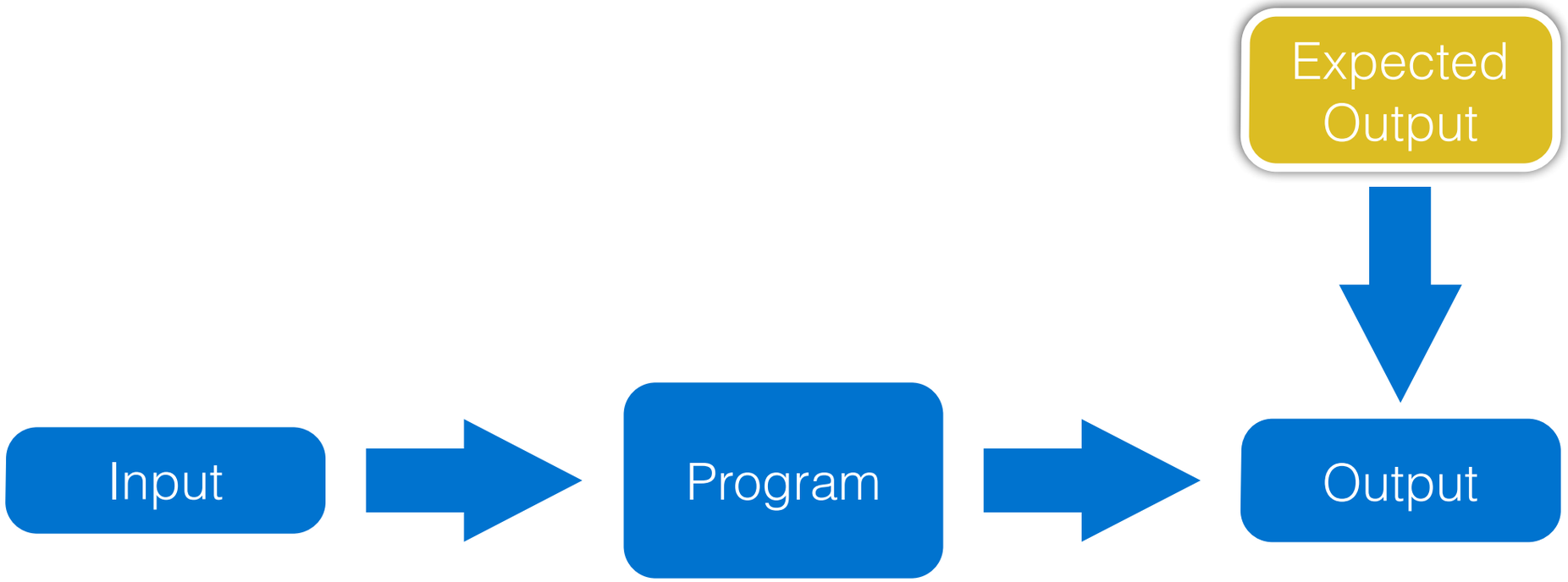
Unit Testing Experiment

José Miguel Rojas, Gordon Fraser - University of Sheffield

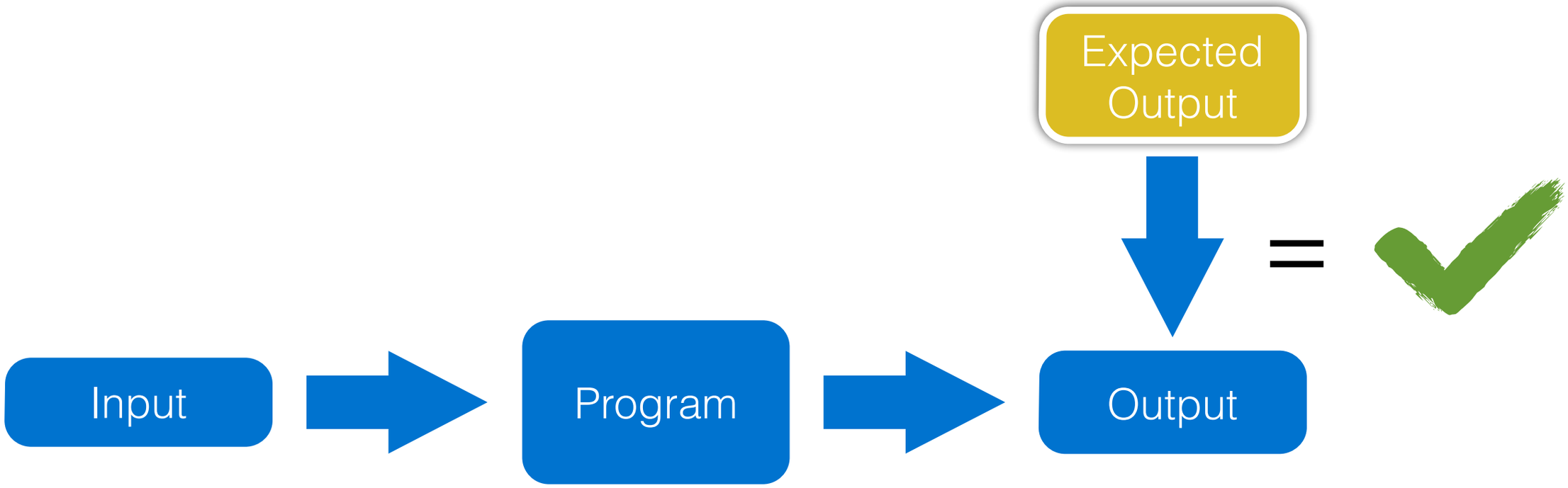
Andrea Arcuri - Simula Research Labs, Norway

Testing

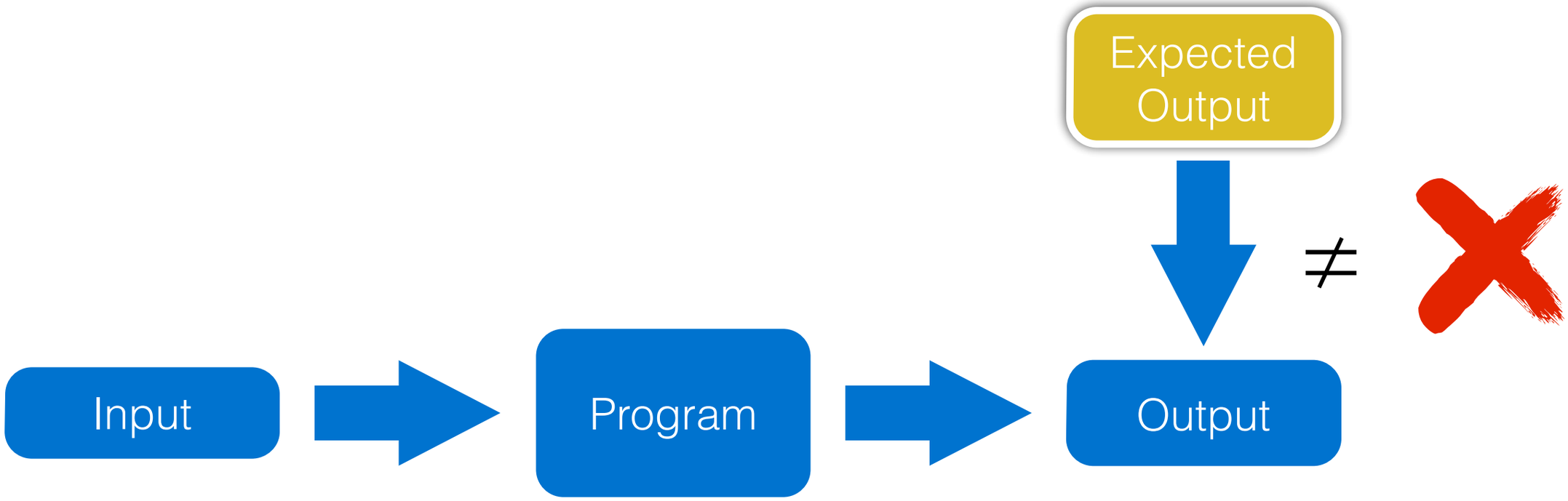
Testing



Testing



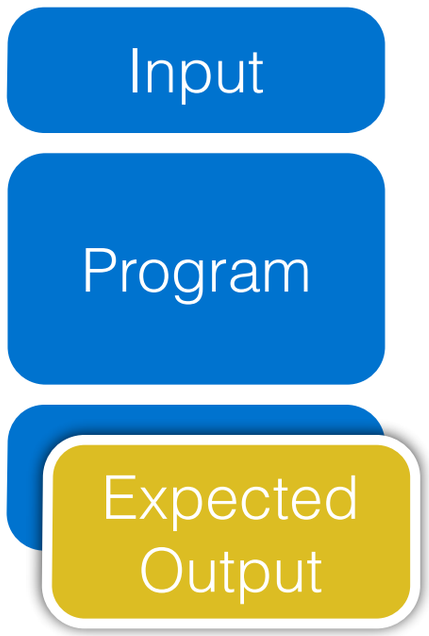
Testing



Unit Testing

```
@Test

public void test()
{
```

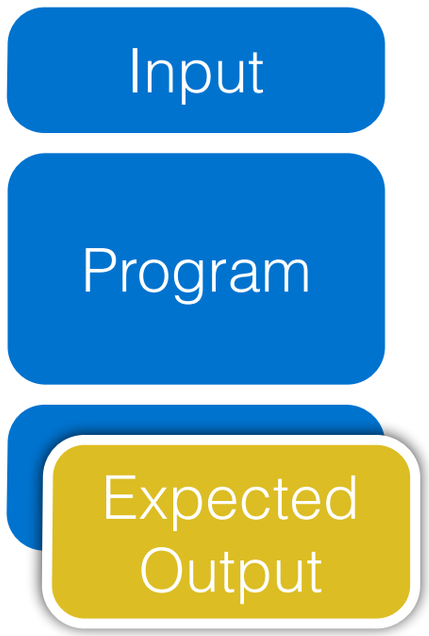



```
}
```

JUnit Testing

```
@Test

public void test()
{
```

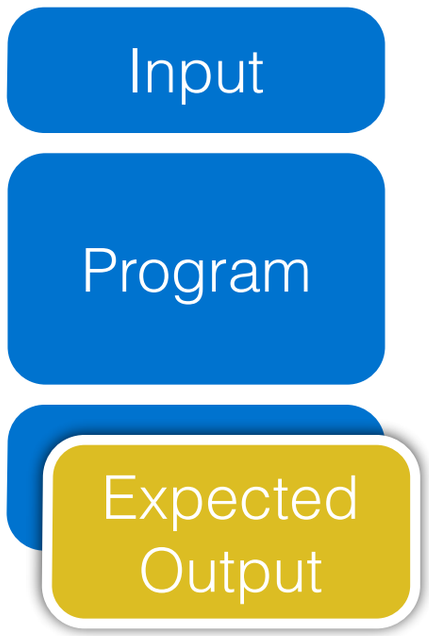



```
}
```

JUnit Testing

```
@Test

public void test()
{
```

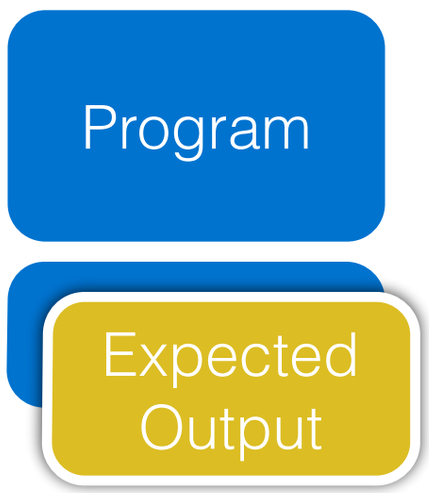



```
}
```

JUnit Testing

```
@Test

public void test()
{
    // Setup
```




```
}
```

JUnit Testing

```
@Test

public void test()
{
        // Setup

        // Exercise

}
```

JUnit Testing

```
@Test

public void test()
{
    // Setup

    // Exercise

    // Check
}
```

JUnit Testing

```
@Test

public void test()
{
    // Setup

    // Exercise

    // Check
}
```

JUnit Testing

```
@Test

public void test()
{
    int x = 2;
    int y = 2;
    // Exercise

    // Check

}
```

JUnit Testing

```
@Test

public void test()
{
    int x = 2;
    int y = 2;

    int result = sum(x, y);

    // Check

}
```

JUnit Testing

```
@Test

public void test()
{
    int x = 2;
    int y = 2;

    int result = sum(x, y);

    assertEquals(4, result);

}
```

JUnit Testing

```
@Test

public void test()
{
    int x = 2;
    int y = 2;

    int result = sum(x, y);

    assertEquals(4, result);

}
```

JUnit Testing

```
@Test

public void test()
{
    int x = 2;
    int y = 2;

    int result = sum(x, y);

    assertEquals(4, result);

}
```

```
@Test

@Ignore
```

JUnit Testing

```
@Test

public void tes
{
    int x = 2;
    int y = 2;

    int resul  = sum(x, y);

    assertEquals(4, result);
}
```

```
@Test

@Ignore
```

```
assertTrue(condition);
assertFalse(condition);

assertNull(condition);
assertNonNull(condition);

assertEquals(expected, actual);
```

Organising Tests

```
public class TestMyClass {
   @Test
   public void test0() {
   }
   @Test
   public void test1() {
   }
}
```

Testing Exceptions

```
@Test
public void test() {
  try {
    foo.bar();
    fail(“Expected exception!”);
  } catch(Exception e) {
    // Expected exception
  }
}
```

Testing Exceptions

```
@Test(expected = Exception.class)
public void test() {
    foo.bar();
}
```

When to test?

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
   */
   public boolean bar(int x) {


   }
}
```

When to test?

```
@Test

public void test1() {

    Foo foo = new Foo();

    assertTrue(foo.bar(0));

}
```

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
    */
    public boolean bar(int x) {


    }

}
```

When to test?

```
@Test
public void test1() {
   Foo foo = new Foo();
   assertTrue(foo.bar(0));
}

@Test
public void test2() {
   Foo foo = new Foo();
   assertFalse(foo.bar(-1));
}
```

```
public class Foo() {
  /**
      Returns true if
      x is non-negative
    */
    public boolean bar(int x) {

    }
}
```

When to test?

```
@Test
public void test1() {
   Foo foo = new Foo();
   assertTrue(foo.bar(0));
}

@Test
public void test2() {
   Foo foo = new Foo();
   assertFalse(foo.bar(-1));
}
```

```
public class Foo() {
  /**
      Returns true if
      x is non-negative
   */
   public boolean bar(int x) {
     return x >= 0;
   }
}
```

When to test?

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
   */
   public boolean bar(int x) {


   }
}
```

When to test?

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
   */
   public boolean bar(int x) {

     return x >= 0;

   }

}
```

When to test?

```
@Test

public void test1() {

    Foo foo = new Foo();

    assertTrue(foo.bar(0));

}
```

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
    */
    public boolean bar(int x) {

      return x >= 0;

    }

}
```

When to test?

```
@Test
public void test1() {
   Foo foo = new Foo();
   assertTrue(foo.bar(0));
}

@Test
public void test2() {
   Foo foo = new Foo();
   assertFalse(foo.bar(-1));
}
```

```
public class Foo() {
  /**
      Returns true if
      x is non-negative
   */
   public boolean bar(int x) {
     return x >= 0;
   }
}
```

Unit Test Generation

Unit Test Generation

- EvoSuite generates JUnit tests automatically

Unit Test Generation

- EvoSuite generates JUnit tests automatically
- These tests will assert the *observed* behaviour

Unit Test Generation

- EvoSuite generates JUnit tests automatically
- These tests will assert the *observed* behaviour
- They will pass on all bugs - assertions need to be checked, removed, added, etc

Unit Test Generation

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
   */
   public boolean bar(int x) {


   }

}
```

Unit Test Generation

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
   */
   public boolean bar(int x) {

     return x <= 0;

   }

}
```

Unit Test Generation

```
@Test

public void test1() {

   Foo foo = new Foo();
   assertTrue(foo.bar(0));
}
```

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
   */
   public boolean bar(int x) {

     return x <= 0;

   }

}
```

Unit Test Generation

```
@Test
public void test1() {
   Foo foo = new Foo();
   assertTrue(foo.bar(0));
}

@Test
public void test2() {
   Foo foo = new Foo();
   assertTrue(foo.bar(-1));
}
```

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
   */
   public boolean bar(int x) {

     return x <= 0;

   }

}
```

Unit Test Generation

```
@Test

public void test1() {

   Foo foo = new Foo();
   assertTrue(foo.bar(0));
}

@Test

public void test2() {

   Foo foo = new Foo();
   assertFalse(foo.bar(-1))
}
```

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
    */
    public boolean bar(int x) {

      return x <= 0;

    }

}
```

Unit Test Generation

```
@Test

public void test1() {

   Foo foo = new Foo();
   assertTrue(foo.bar(0));
}

@Test

public void test2() {

   Foo foo = new Foo();
   assertFalse(foo.bar(-1))
}
```

```
public class Foo() {

  /**
      Returns true if
      x is non-negative
   */
   public boolean bar(int x) {
     return x >= 0;
   }
}
```

The Experiment

Objective:

The Experiment

Objective: Implement the target class and a test suite

The Experiment

Objective: Implement the target class and a test suite

The Experiment

Objective: Implement the target class and a test suite

- Implement all methods according to their JavaDoc specification

The Experiment

Objective: Implement the target class and a test suite

- Implement all methods according to their JavaDoc specification
- Write JUnit tests to cover all code

The Experiment

Objective: Implement the target class and a test suite

- Implement all methods according to their JavaDoc specification
- Write JUnit tests to cover all code
- You can use EvoSuite to generate tests

The Experiment

Objective: Implement the target class and a test suite

- Implement all methods according to their JavaDoc specification
- Write JUnit tests to cover all code
- You can use EvoSuite to generate tests
- All tests should pass

The Experiment

Objective: Implement the target class and a test suite

- Implement all methods according to their JavaDoc specification
- Write JUnit tests to cover all code
- You can use EvoSuite to generate tests
- All tests should pass
- Tests for unfinished/wrong behaviour may fail

The Experiment

The Experiment

- You are free to use EclEmma to measure code coverage of your tests

The Experiment

- You are free to use EclEmma to measure code coverage of your tests
- Please don’t install any plugins in Eclipse

The Experiment

- You are free to use EclEmma to measure code coverage of your tests
- Please don’t install any plugins in Eclipse
- Please don’t change the build process

The Experiment

- You are free to use EclEmma to measure code coverage of your tests
- Please don’t install any plugins in Eclipse
- Please don’t change the build process
- Please do not place tests in any other class

The Experiment

- You are free to use EclEmma to measure code coverage of your tests
- Please don’t install any plugins in Eclipse
- Please don’t change the build process
- Please do not place tests in any other class
- Please don’t edit any other class

The Experiment

- You are free to use EclEmma to measure code coverage of your tests
- Please don't install any plugins in Eclipse
- Please don't change the build process
- Please do not place tests in any other class
- Please don't edit any other class
- Exam conditions
  No conferring, no use of mobiles

Think-aloud Session

- Verbalise your thoughts
  - What are you thinking?
  - What are you doing?
    - Why did you decide to do that?
    - Do you like something? Do you miss anything?
- Audio and screen recording
- Keep talking, it tends to become natural

Target Class: PredicatedMap

- Package collections.map
- Decorates a map object to validate that additions match a predicate represented by a Predicate object
- Examples
  - A PredicatedMap created with a NoNullPredicate would ensure that no null keys are added to the map
  - A PredicatedMap created with a PositivePredicate would ensure that only positive integers are added as keys to the map

Target Class: PredicatedMap

- Maps
  - A map models a searchable collection of key-value entries
  - Multiple entries with the same key are not allowed
  - Main operations: get(k), put(k,v), entrySet(), iterator(), keySet(), size(), isEmpty()

Target Class: PredicatedMap

- Example of Map behaviour

Operation	Output	Map
new PredicatedMap()		{}
isEmpty()	TRUE	{}
put(“Earth”,1)	null	{(“Earth”,1)}
put(“Mars”,0)	null	{(“Earth”,1),(“Mars”,0)}
put(“Mars”,2)	0	{(“Earth”,1),(“Mars”,2)}
size()	2	{(“Earth”,1),(“Mars”,2)}
get(“Mars”)	2	{(“Earth”,1),(“Mars”,2)}
get(“Uranus”)	null	{(“Earth”,1),(“Mars”,2)}
isEmpty()	FALSE	{(“Earth”,1),(“Mars”,2)}

Target Class: FilterIterator

- Package collections.iterators
- Implements a java.util.Iterator for any Java Collection
  - An Iterator abstracts the process of scanning through a collection of elements
  - An Iterator allows for a linear traversal of the collection, through pre-defined interfaces: hasNext(), next()
- Only those elements that satisfy a given Predicate evaluation are iterated over

Target Class: FilterIterator

- Example

Target Class: FilterIterator

- Example

Target Class: FilterIterator

- Example

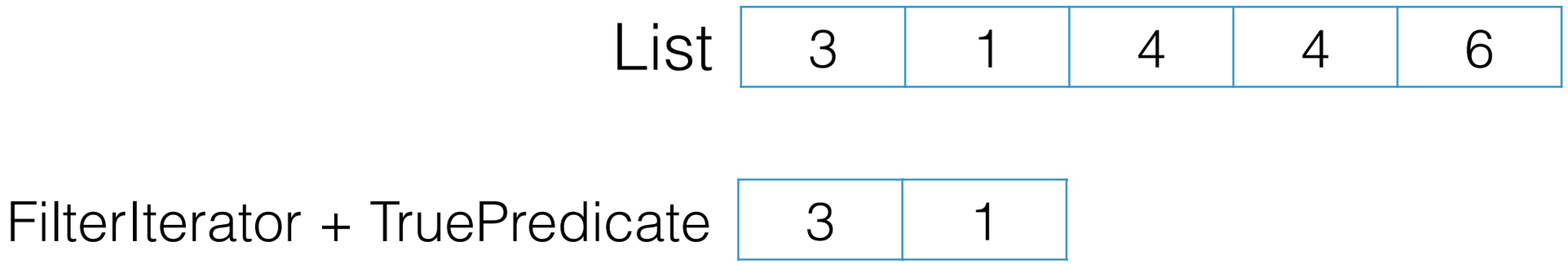
FilterIterator + TruePredicate

Target Class: FilterIterator

- Example

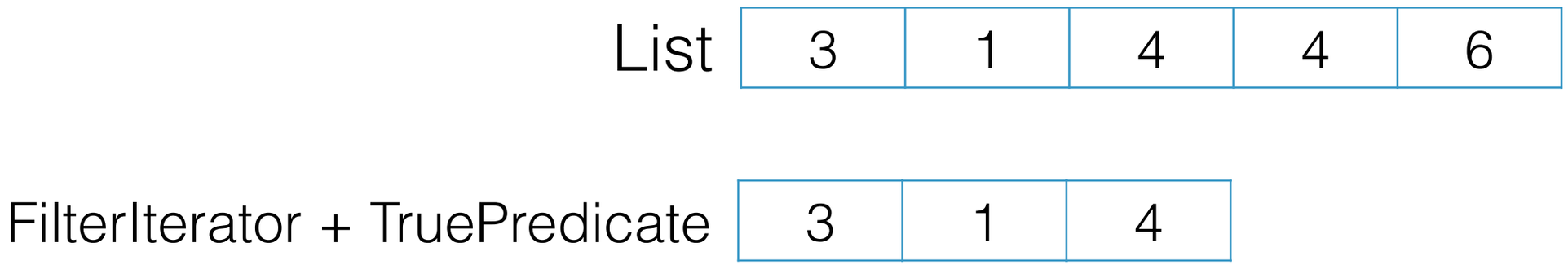
Target Class: FilterIterator

- Example



Target Class: FilterIterator

- Example



Target Class: FilterIterator

- Example

List	3	1	4	4	6

FilterIterator + TruePredicate	3	1	4	4

Target Class: FilterIterator

- Example

List	3	1	4	4	6

FilterIterator + TruePredicate	3	1	4	4	6

Target Class: FilterIterator

- Example

List

3	1	4	4	6

FilterIterator + TruePredicate

3	1	4	4	6

FilterIterator + UniquePredicate

Target Class: FilterIterator

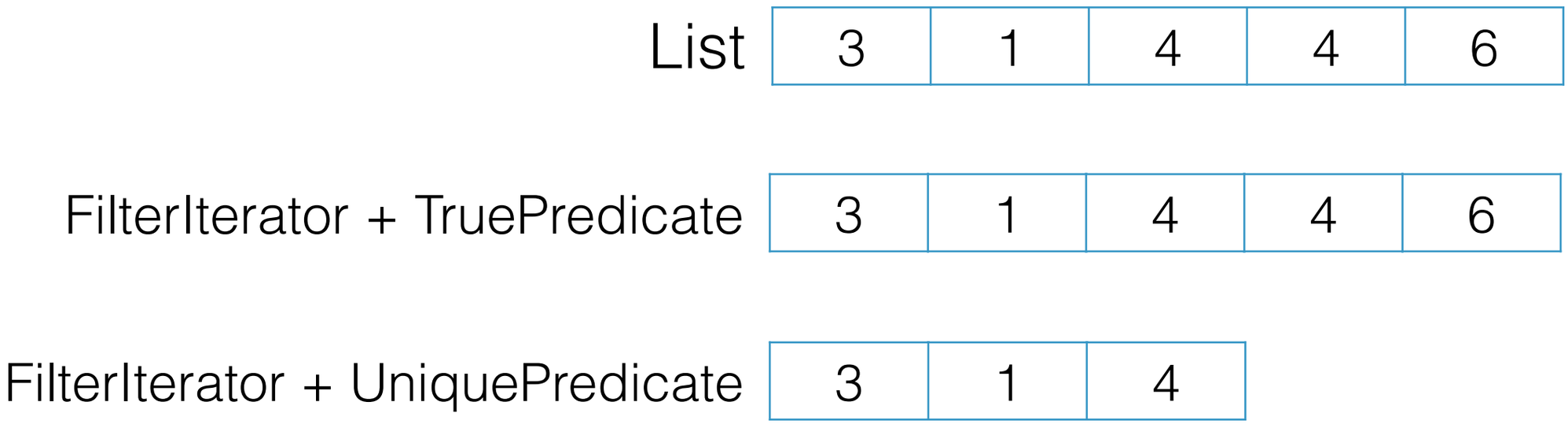
- Example

Target Class: FilterIterator

- Example

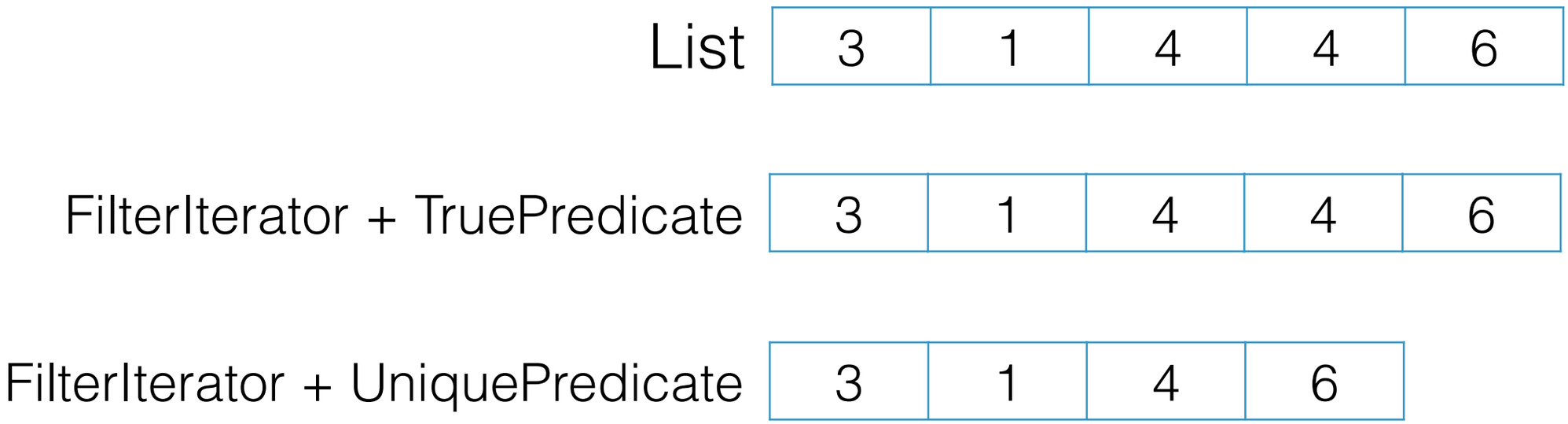
Target Class: FilterIterator

- Example



Target Class: FilterIterator

- Example



Target Class: FilterIterator

- Example

List

3	1	4	4	6

FilterIterator + TruePredicate

3	1	4	4	6

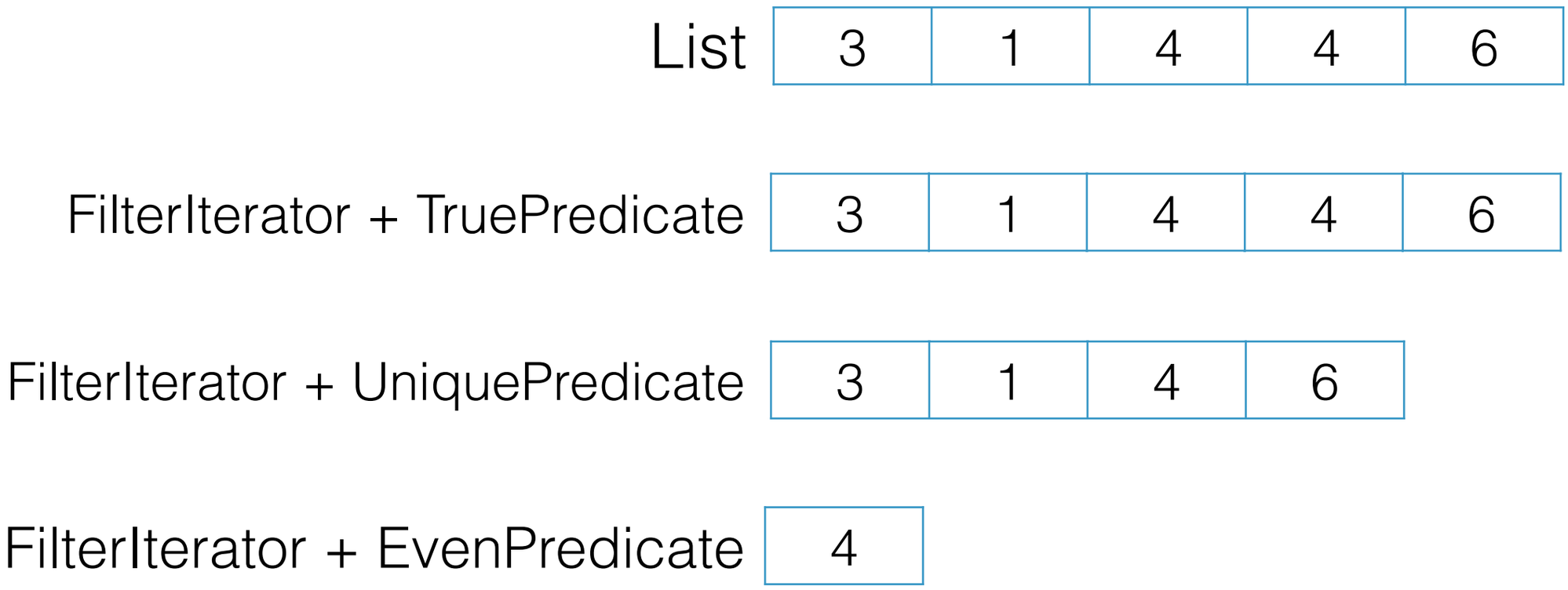
FilterIterator + UniquePredicate

3	1	4	6

FilterIterator + EvenPredicate

Target Class: FilterIterator

- Example



Target Class: FilterIterator

- Example

List

3	1	4	4	6

FilterIterator + TruePredicate

3	1	4	4	6

FilterIterator + UniquePredicate

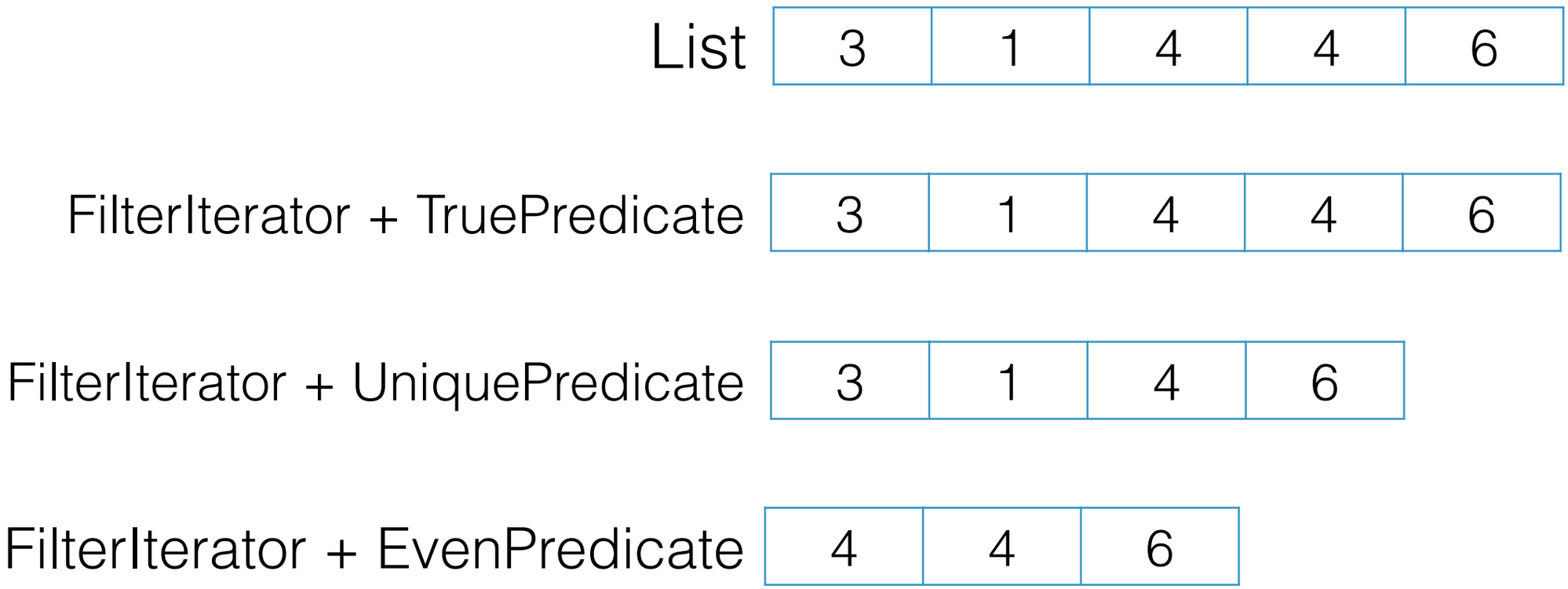
3	1	4	6

FilterIterator + EvenPredicate

4	4

Target Class: FilterIterator

- Example



Target Class: FilterIterator

- Traditional use of an Iterator object

```
// obtain an iterator from an iterable collection (Set, List,…)
Iterator it = collection.iterator();

// iterate while there are elements ahead of the cursor
while (it.hasNext()) {
      // fetch next element
      Object obj = it.next();

      // do something with the element
}
```

Target Class: ListPopulation

- Package math.genetics. Implements math.genetics.Population interface
- Represents a population of chromosomes as a List of Chromosome objects
- *Abstract class*: cannot be instantiated directly
  - Dependency classes, e.g., Chromosome, are also abstract
  - Concrete subclasses must be used for testing
  - ElitisticListPopulation is a concrete subclass of ListPopulation; DummyBinaryChromosome and DummyListChromosome are concrete subclasses of Chromosome

Target Class: ListPopulation

- Instantiating Chromosome: DummyListChromosome, or…

```
Chromosome c1 = new Chromosome() {
    public double fitness() {
        return 15;
    }
};
```

- Instantiating ListPopulation: ElitisticListPopulation, or…

```
ArrayList<Chromosome> chromosomes = new ArrayList<Chromosome> ();
chromosomes.add(c1);
ListPopulation population = new ListPopulation(chromosomes, 10) {
    public Population nextGeneration() {
        return null;                    // not important
    }
};
```

Target Class: FixedOrderComparator

- Package collections.comparators
- Implements a java.util.Comparator which imposes a specific order on a specific set of Objects, presented as an argument to the constructor
- Method compare(Object,Object) compares two objects according to the specified order
- In general, a Comparator encapsulates the action of comparing two objects according to a given total order relation

Target Class: FixedOrderComparator

- Comparator.compare(Object x, Object y) returns:
  - A negative integer if x comes before y; e.g., if x is less than y
  - Zero if x and y are equal
  - A positive integer if x comes after y; e.g., if x is greater than y
- FixedOrderComparator must store locally the set of elements that it will compare. Choosing a suitable data structure is part of your task

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

```
String[] planets = {"Mercury", "Venus", "Mars"};
FixedOrderComparator distanceFromSun = new FixedOrderComparator(planets);
String[] planets2 = {"Mercury", "Venus", "Mars", "Earth"};
distanceFromSun.addAsEqual(“Venus”, “Earth”);
Arrays.sort(planets2);                          // Sort to alphabetical order
{"Earth", "Mars", "Mercury", "Venus"}
Arrays.sort(planets2, distanceFromSun);    // Back to original order
{"Mercury", "Venus", "Earth", "Mars"}
```

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map
new FixedOrderComparator()		{}

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map
new FixedOrderComparator()		{}
add(“Mercury”)	TRUE	{(“Mercury”,0)}

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map
new FixedOrderComparator()		{}
add(“Mercury”)	TRUE	{(“Mercury”,0)}
add(“Venus”)	TRUE	{(“Mercury”,0),(“Venus”,1)}

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map
new FixedOrderComparator()		{}
add(“Mercury”)	TRUE	{(“Mercury”,0)}
add(“Venus”)	TRUE	{(“Mercury”,0),(“Venus”,1)}
addAsEqual(“Venus”, “Earth”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1)}

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map
new FixedOrderComparator()		{}
add(“Mercury”)	TRUE	{(“Mercury”,0)}
add(“Venus”)	TRUE	{(“Mercury”,0),(“Venus”,1)}
addAsEqual(“Venus”, “Earth”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1)}
add(“Mars”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1),(“Mars”,2)}

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map
new FixedOrderComparator()		{}
add(“Mercury”)	TRUE	{(“Mercury”,0)}
add(“Venus”)	TRUE	{(“Mercury”,0),(“Venus”,1)}
addAsEqual(“Venus”, “Earth”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1)}
add(“Mars”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1),(“Mars”,2)}
compare(“Mercury”, “Earth”)	-1	

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map
new FixedOrderComparator()		{}
add(“Mercury”)	TRUE	{(“Mercury”,0)}
add(“Venus”)	TRUE	{(“Mercury”,0),(“Venus”,1)}
addAsEqual(“Venus”, “Earth”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1)}
add(“Mars”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1),(“Mars”,2)}
compare(“Mercury”, “Earth”)	-1	
compare(“Mars”, “Venus”)	1	

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map
new FixedOrderComparator()		{}
add(“Mercury”)	TRUE	{(“Mercury”,0)}
add(“Venus”)	TRUE	{(“Mercury”,0),(“Venus”,1)}
addAsEqual(“Venus”, “Earth”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1)}
add(“Mars”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1),(“Mars”,2)}
compare(“Mercury”, “Earth”)	-1	
compare(“Mars”, “Venus”)	1	
compare(“Venus”, “Earth”)	0	

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map
new FixedOrderComparator()		{}
add(“Mercury”)	TRUE	{(“Mercury”,0)}
add(“Venus”)	TRUE	{(“Mercury”,0),(“Venus”,1)}
addAsEqual(“Venus”, “Earth”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1)}
add(“Mars”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1),(“Mars”,2)}
compare(“Mercury”, “Earth”)	-1	
compare(“Mars”, “Venus”)	1	
compare(“Venus”, “Earth”)	0	
compare(“Mars”, “Uranus”)	-1 OR 1 OR exc	

Target Class: FixedOrderComparator

- Example of FixedOrderComparator

Operation	Output	Map
new FixedOrderComparator()		{}
add(“Mercury”)	TRUE	{(“Mercury”,0)}
add(“Venus”)	TRUE	{(“Mercury”,0),(“Venus”,1)}
addAsEqual(“Venus”, “Earth”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1)}
add(“Mars”)	TRUE	{(“Mercury”,0),(“Venus”,1), (“Earth”,1),(“Mars”,2)}
compare(“Mercury”, “Earth”)	-1	
compare(“Mars”, “Venus”)	1	
compare(“Venus”, “Earth”)	0	
compare(“Mars”, “Uranus”)	-1 OR 1 OR exc	

Depends on setUnknownObjectBehavior(int)

Target Class: FixedOrderComparator

- A map is an example of suitable data structure (feel free to use any other)

- A map models a searchable collection of key-value entries

- Multiple entries with the same key are not allowed

- Main operations: get(k), put(k,v), entrySet(), iterator(), keySet(), size(), isEmpty()

Target Class: FixedOrderComparator

- Example of Map behaviour

Operation	Output	Map
new HashMap()		{}
isEmpty()	TRUE	{}
put(“Earth”,1)	null	{(“Earth”,1)}
put(“Mars”,0)	null	{(“Earth”,1),(“Mars”,0)}
put(“Mars”,2)	0	{(“Earth”,1),(“Mars”,2)}
size()	2	{(“Earth”,1),(“Mars”,2)}
get(“Mars”)	2	{(“Earth”,1),(“Mars”,2)}
get(“Uranus”)	null	{(“Earth”,1),(“Mars”,2)}
isEmpty()	FALSE	{(“Earth”,1),(“Mars”,2)}