Programación Orientada a Objetos

Colecciones

CEIS

2020-2

Agenda

Agenda

Conceptos

En general

Un colección es un tipo especial de datos usado para almacenar y organizar otros datos

¿Qué colecciones conocen?

- PIMB - PIMO
- LCAL - MDIS

¿Cómo las categorizamos?

Conceptos

Operaciones-básicas

Operaciones-analizadoras

Conceptos

Operaciones-básicas

- Crear
- Adicionar un elemento a la colección
- Eliminar un elemento de la colección

Operaciones-analizadoras

- ¿Cuántos elementos hay en la colección?
- ¿Qué elemento está en una posición de la colección?

- ¿Está un elemento en la colección?

Conceptos

En POOB

Un colección es un tipo especial de objetos usado para almacenar y organizar referencias a otros objetos

¿Qué colecciones hemos manejado?

Agenda

Laboratorios

Laboratorio 2

```
import java.util.Stack;

/** Calculadora.java
 * Representa una calculadora de conjuntos
 * @author ESCUELA 2020-2
 */

public class CalConjuntos{

    private Stack<Conjunto> operandos;
    //Consultar en el API Java la clase Stack


public class Conjunto{


    /**
     * Constructor del conjunto
     */

    public Conjunto () {
    }

     /**
     * Constructor del conjunto
     */
    public Conjunto(String [] elementos){
    }
```

Laboratorios

Laboratorio 3

```
public class Isla{
    public static final int MAXIMO = 500;
    private static Isla isla = null;

    public static Isla demeIsla() {
        if (isla==null){
            isla=new Isla();
        }
        return isla;
    }

    private static void nuevaIsla() {
        isla=new Isla();
    }

    public static void cambieIsla(Isla d) {
        isla=d;
    }

    private ArrayList<EnIsla> elementos;
    private int tesoroPosX;
    private int tesoroPosY;
    private boolean encontraronTesoro;
```

¿Qué contienen? ¿Qué permiten?

Laboratorios

Laboratorio 4

```
import java.util.LinkedList;
import java.util.ArrayList;

//No olvide documentar
public class Iemois{
    private LinkedList <Mooc> cursos;



    public Iemois(){
        cursos = new LinkedList<Mooc>();
    }
```

¿Qué contienen? ¿Qué permiten?

Agenda

Conceptos

Tipos básicos

1. Colecciones simples

2. Colecciones con clave

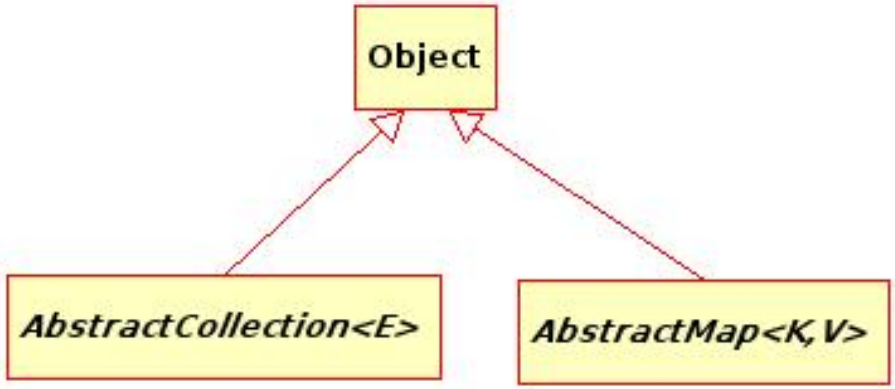
Conceptos

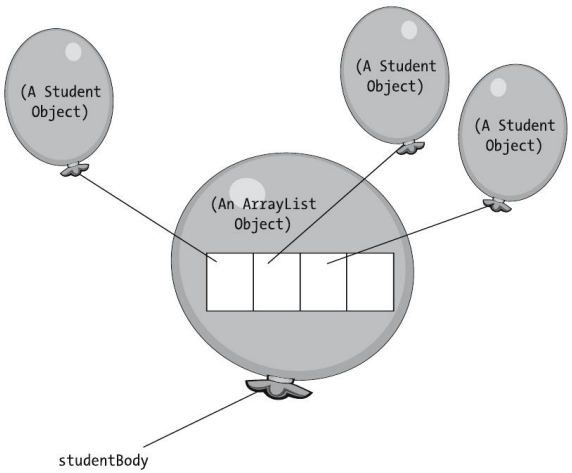
Tipos básicos

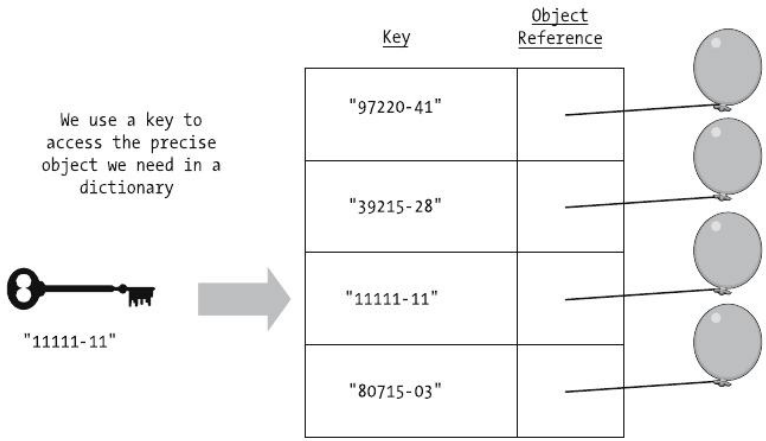
1. Colecciones simples
   - Pueden existir elementos repetidos, interesa la posición.
   - No pueden existir elemento repetidos
2. Colecciones con clave
   - No pueden existir claves repetidas

Contexto java

Dos objetos base

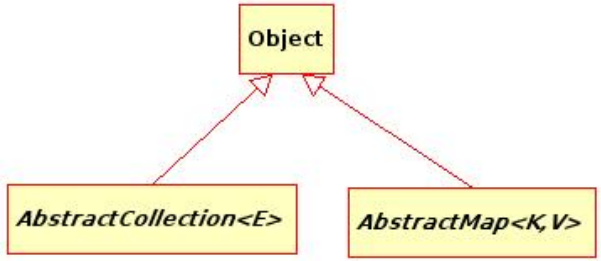




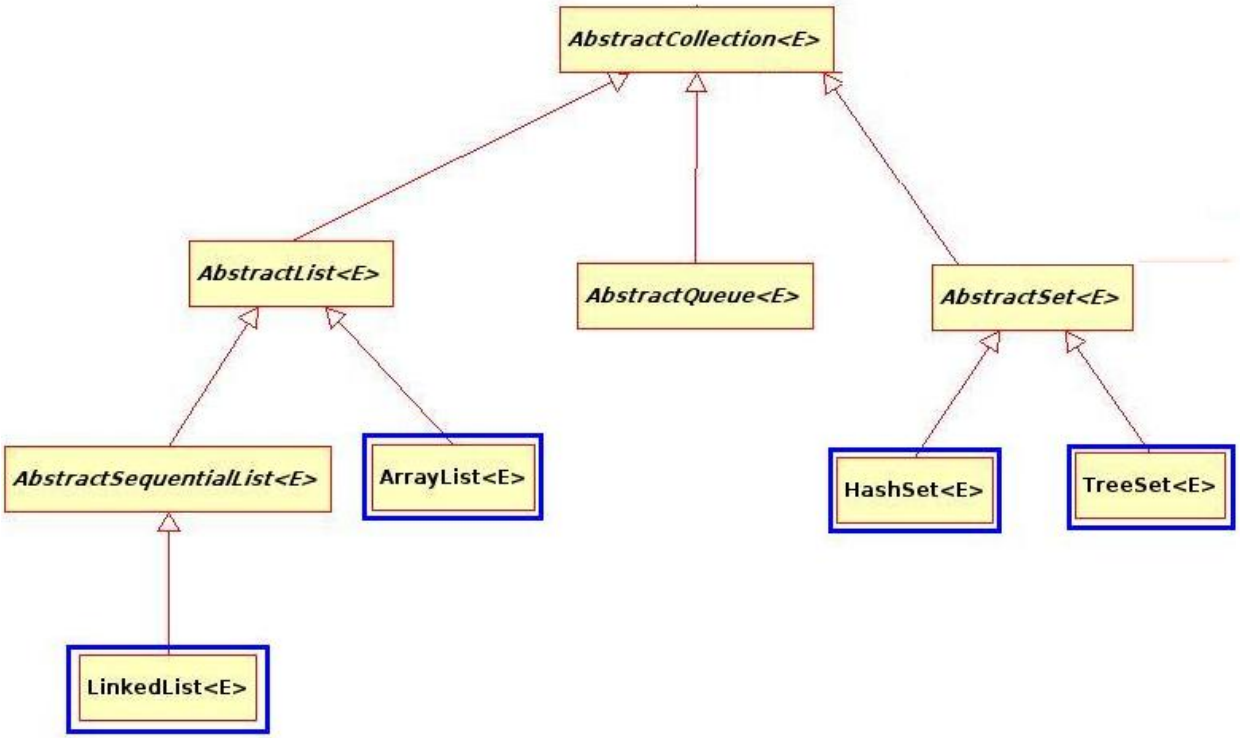


Contexto java

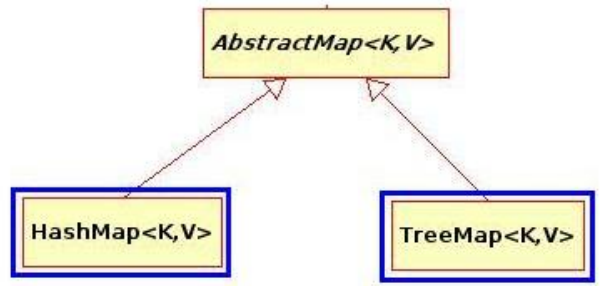
Dos objetos base



Sin clave



Con clave



En java

Tipos básicos

1. Colecciones simples
   - Pueden existir elementos repetidos.
   - No pueden existir elemento repetidos
2. Colecciones con clave

En java

Tipos básicos

1. Colecciones simples
   - Pueden existir elementos repetidos.
     `List`: `ArrayList`, `LinkedList`
   - No pueden existir elemento repetidos

     `Set`:`HashSet`, `TreeSet`
2. Colecciones con clave

   `Map`:`HashMap`, `TreeMap`

Agenda

Representación

Diferenciadores

1. `Array`

   `ArrayList`
2. `Linked`

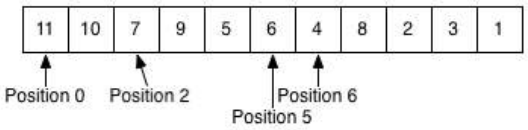
   `LinkedList`
3. `Hash`

   `HashSet`, `HashMap`
4. `Tree`

   `TreeSet`, `TreeMap`

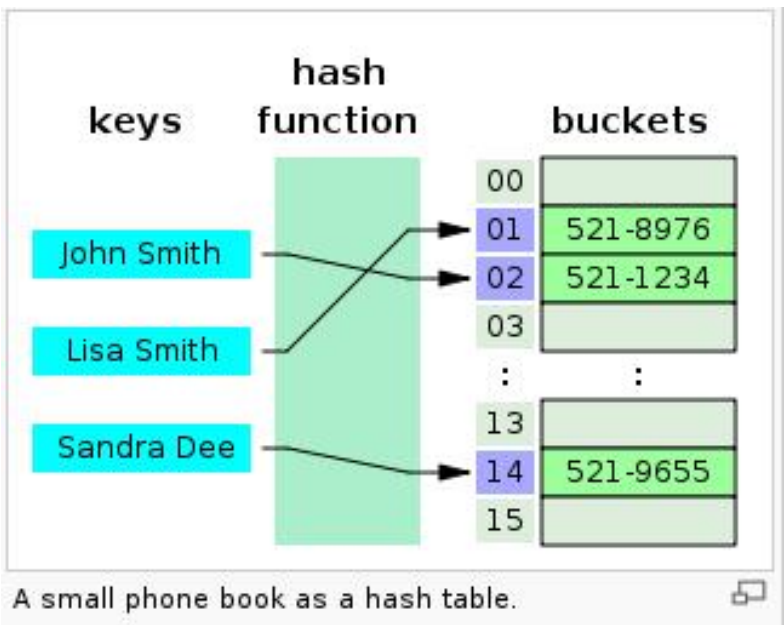
Representación

Array



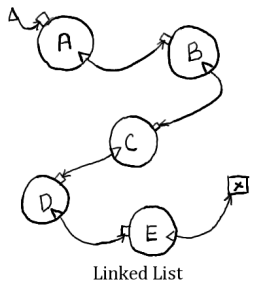
Arreglo que puede cambiar de tamaño.

Hash



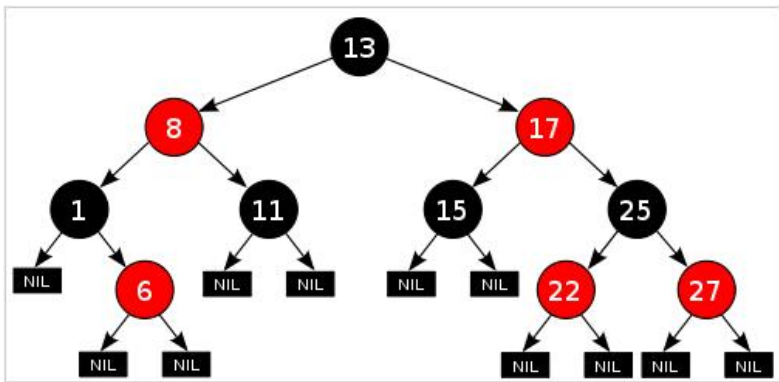
A small phone book as a hash table.

List



Lista de apuntadores.

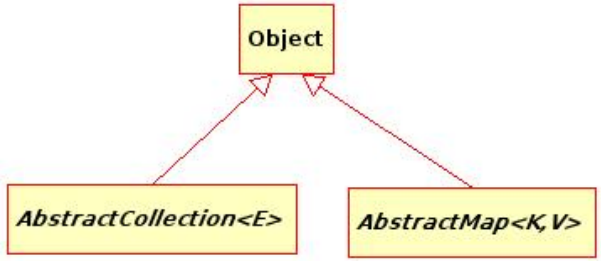
Tree



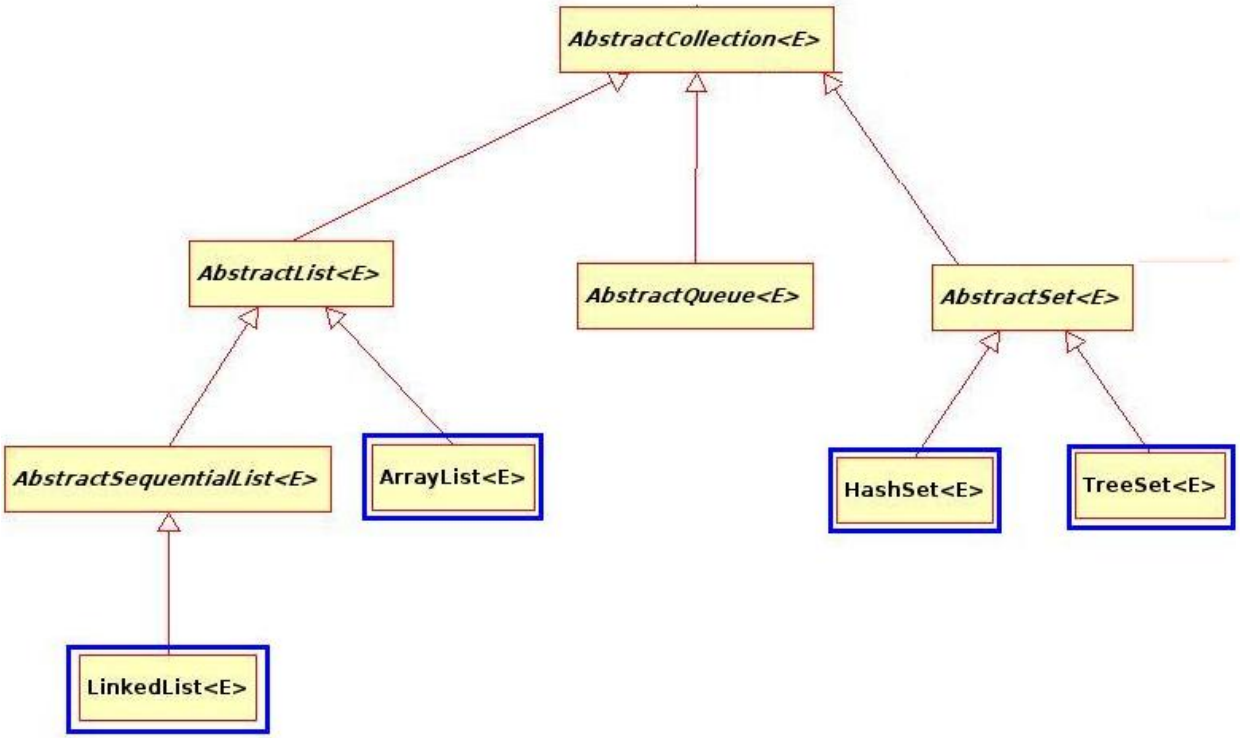
Árbol rojo-negro (ordenado - balanceado).

Contexto java

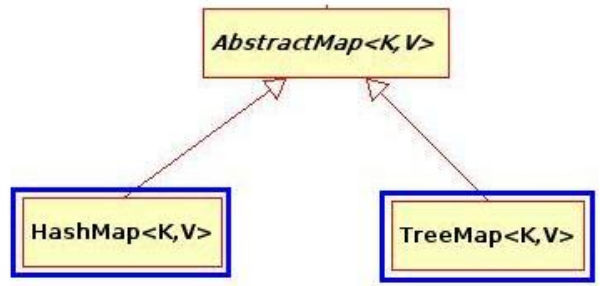
Dos objetos base



Sin clave

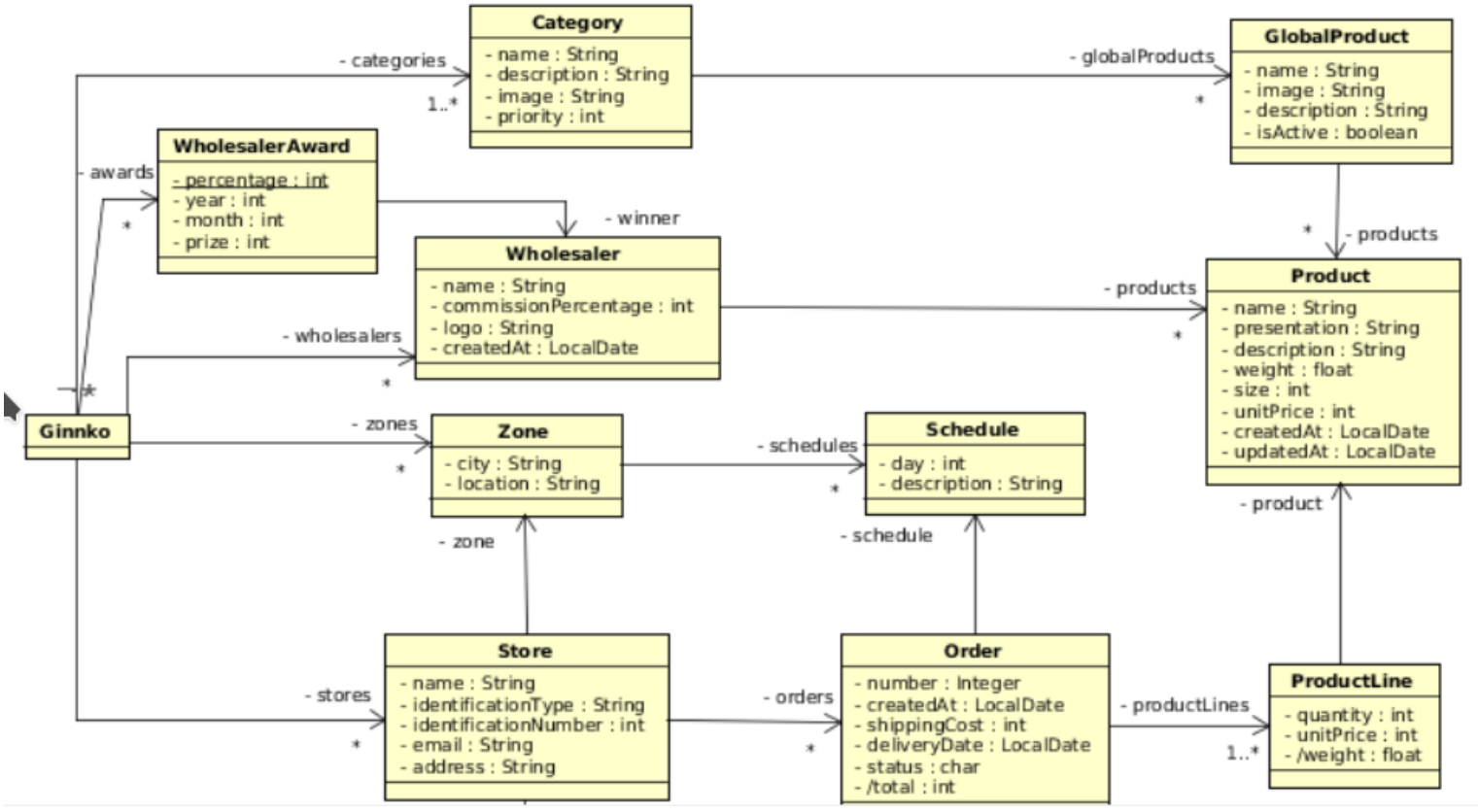


Con clave



Agenda

Parcial



Diseñando

- Los distribuidores los queremos consultar ordenados por nombre
- Las tiendas las queremos consultar por nombres y por tipo y número de identificación
- Las ordenes las queremos consultar desde Ginnko ordenadas por estado

Agenda

Conceptos

Operaciones-básicas

- Crear
- Adicionar un objeto a la colección
- Eliminar un objeto de la colección

Operaciones-analizadoras

- ¿Cuántos elementos hay en la colección?
- ¿Qué elemento está en una posición de la colección?

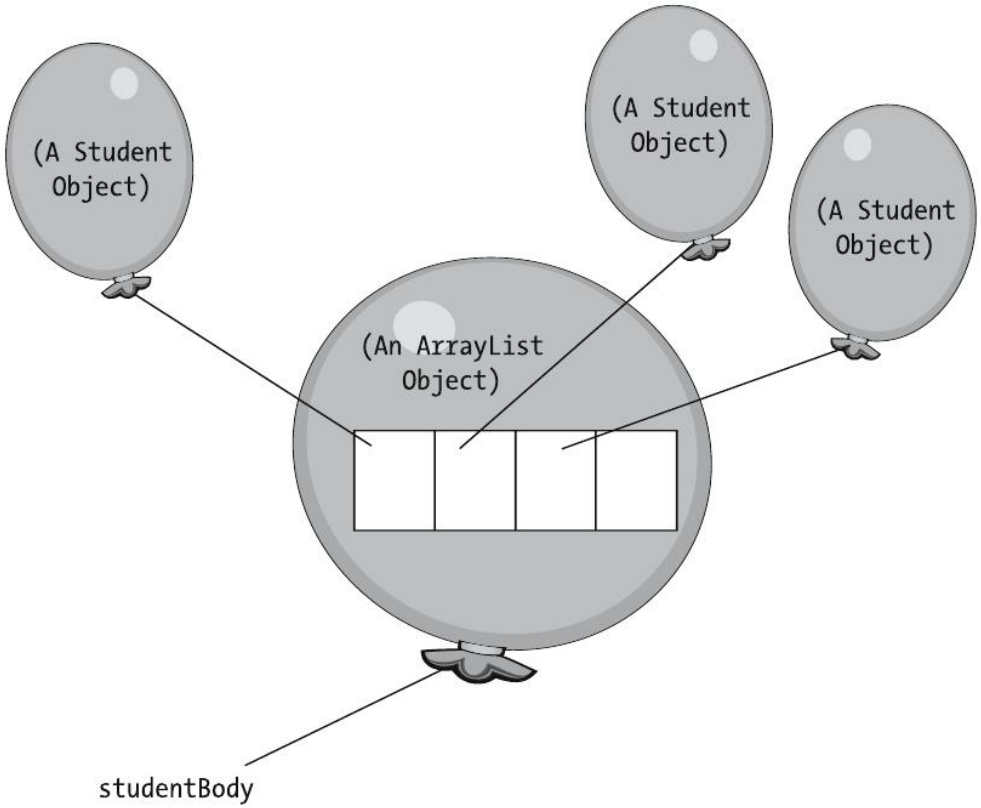
- ¿Está un elemento específico en la colección?

En general

Creando

```
ArrayList<Student> studentBody ;  // ArrayList is one of Java's predefined collec
studentBody  = new ArrayList<Student>();
```

En Uso



Conceptos

Operaciones básicas

- Crear
- Adicionar un objeto a la colección
- Eliminar un objeto de la colección

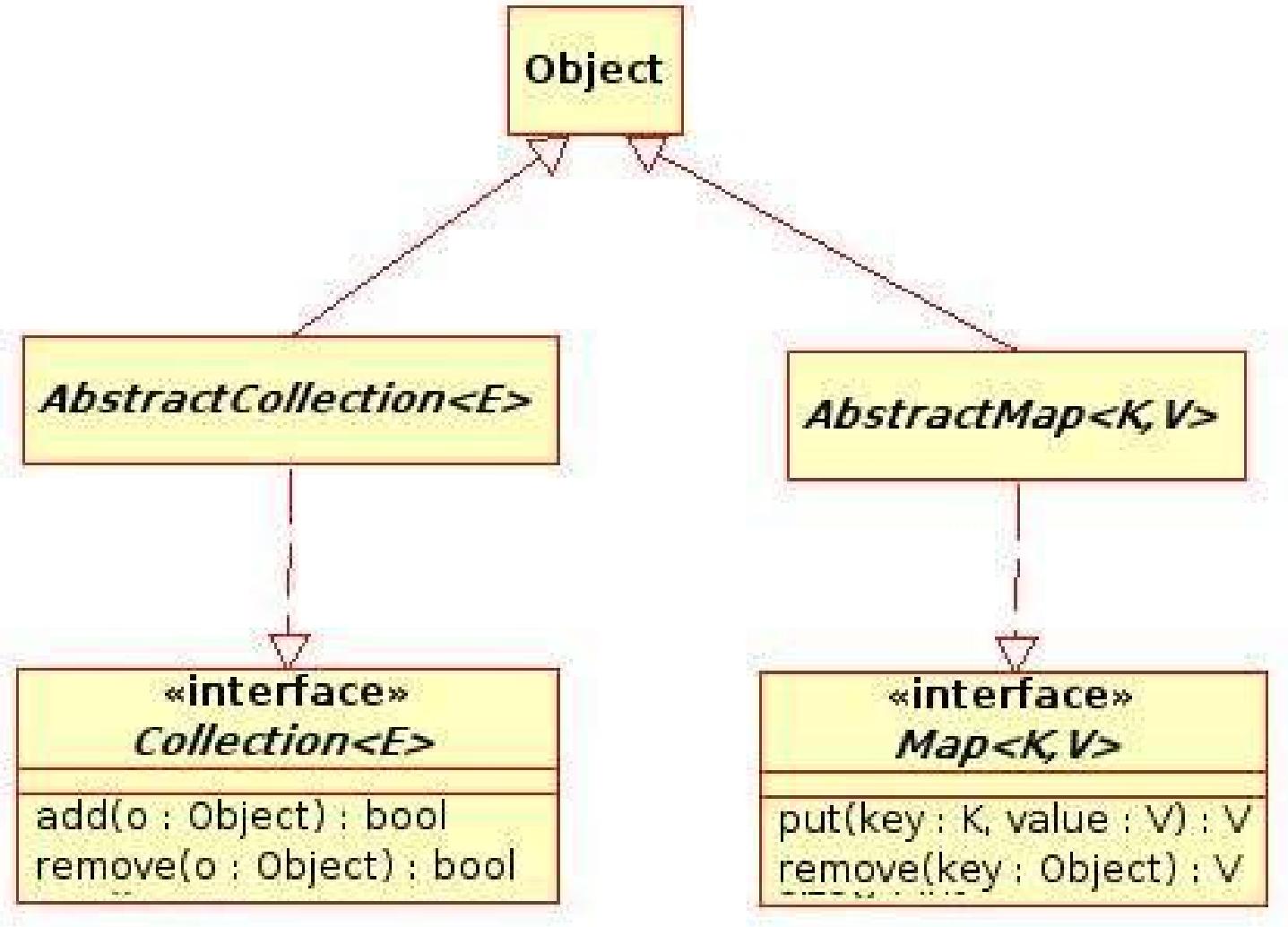
Operaciones-analizadoras

- ¿Cuántos elementos hay en la colección?
- ¿Qué elemento está en una posición de la colección?

- ¿Está un elemento en la colección?

Contexto java

Operaciones básicas



Agenda

Conceptos

Operaciones básicas

- Crear
- Adicionar un objeto a la colección
- Eliminar un objeto de la colección

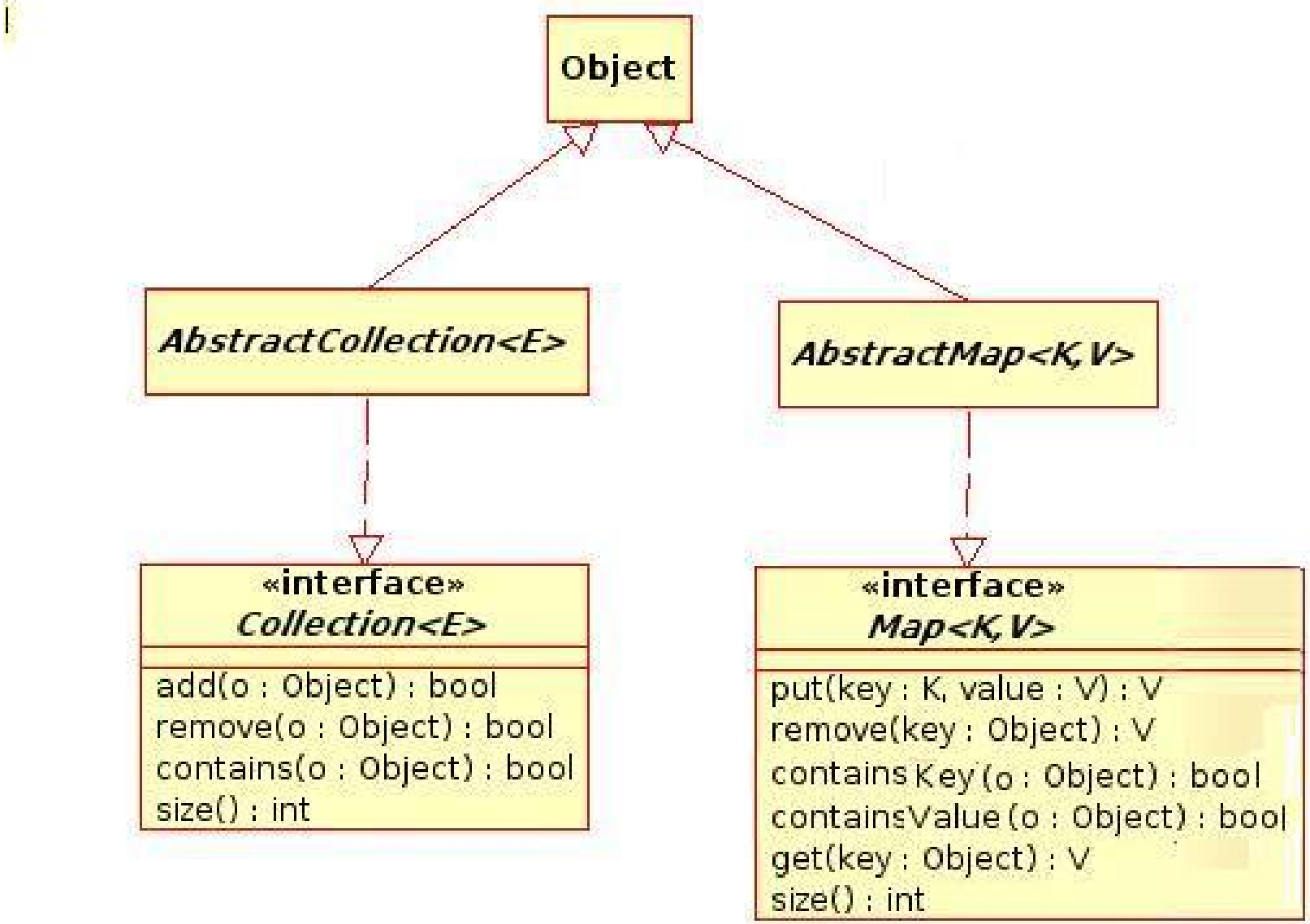
Operaciones-analizadoras

- ¿Cuántos elementos hay en la colección?
- ¿Qué elemento está en una posición de la colección?

- ¿Está un elemento een la colección?

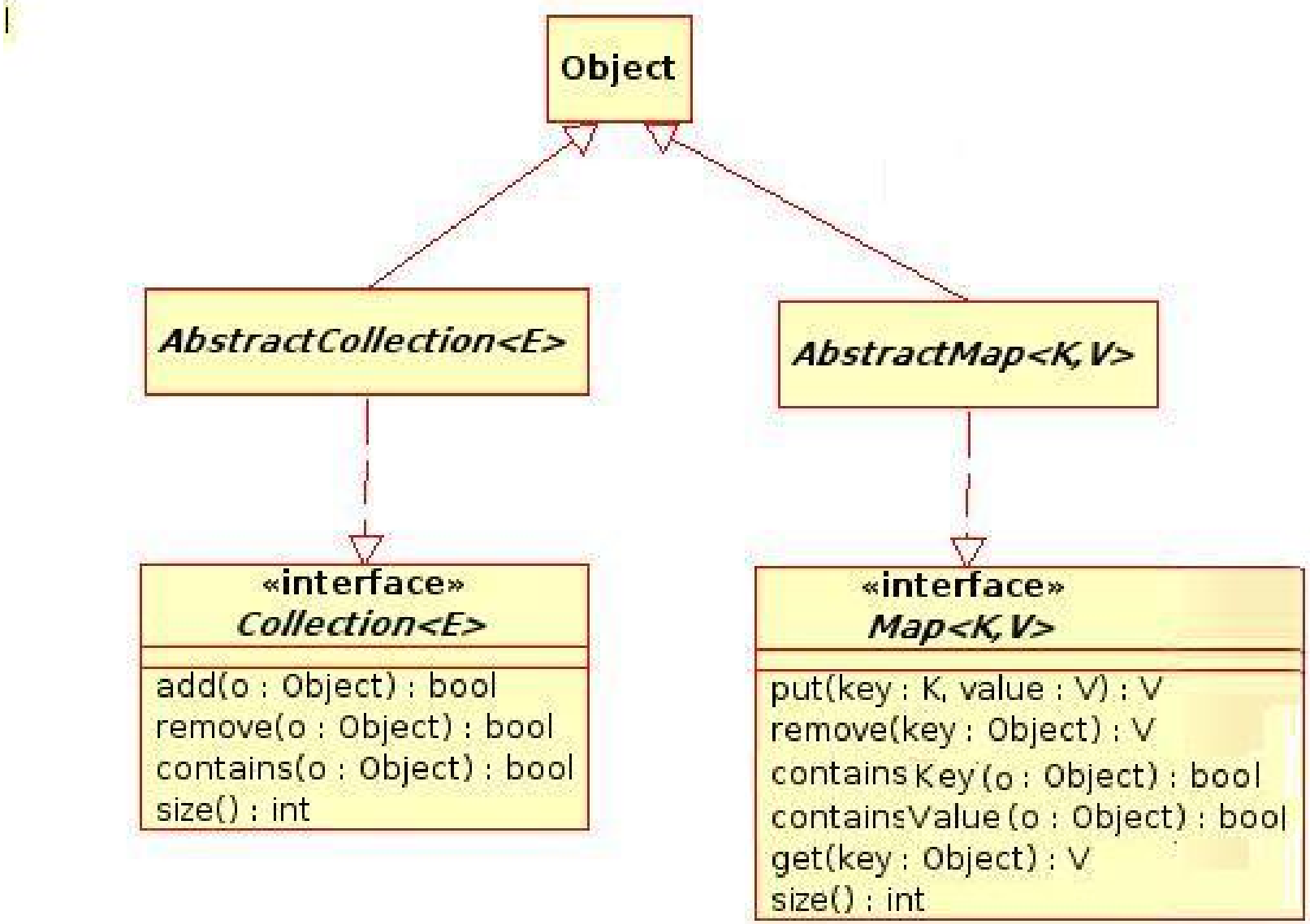
Contexto java

Operaciones básicas



Contexto java

Operaciones básicas



Conceptos

Operaciones básicas

- Crear
- Adicionar un objeto a la colección
- Eliminar un objeto de la colección

Operaciones-analizadoras

- ¿Cuántos elementos hay en la colección?
- ¿Qué elemento está en una posición de la colección?

- ¿Está un elemento específico en la colección?
-
- ¿Cuáles son cada uno de los objetos que hay en la colección?

Contexto java

Recorriendo

```
for (type referenceVariable : collectionName) {
    // Pseudocode.
    manipulate the referenceVariable as desired
}
```

Especificación

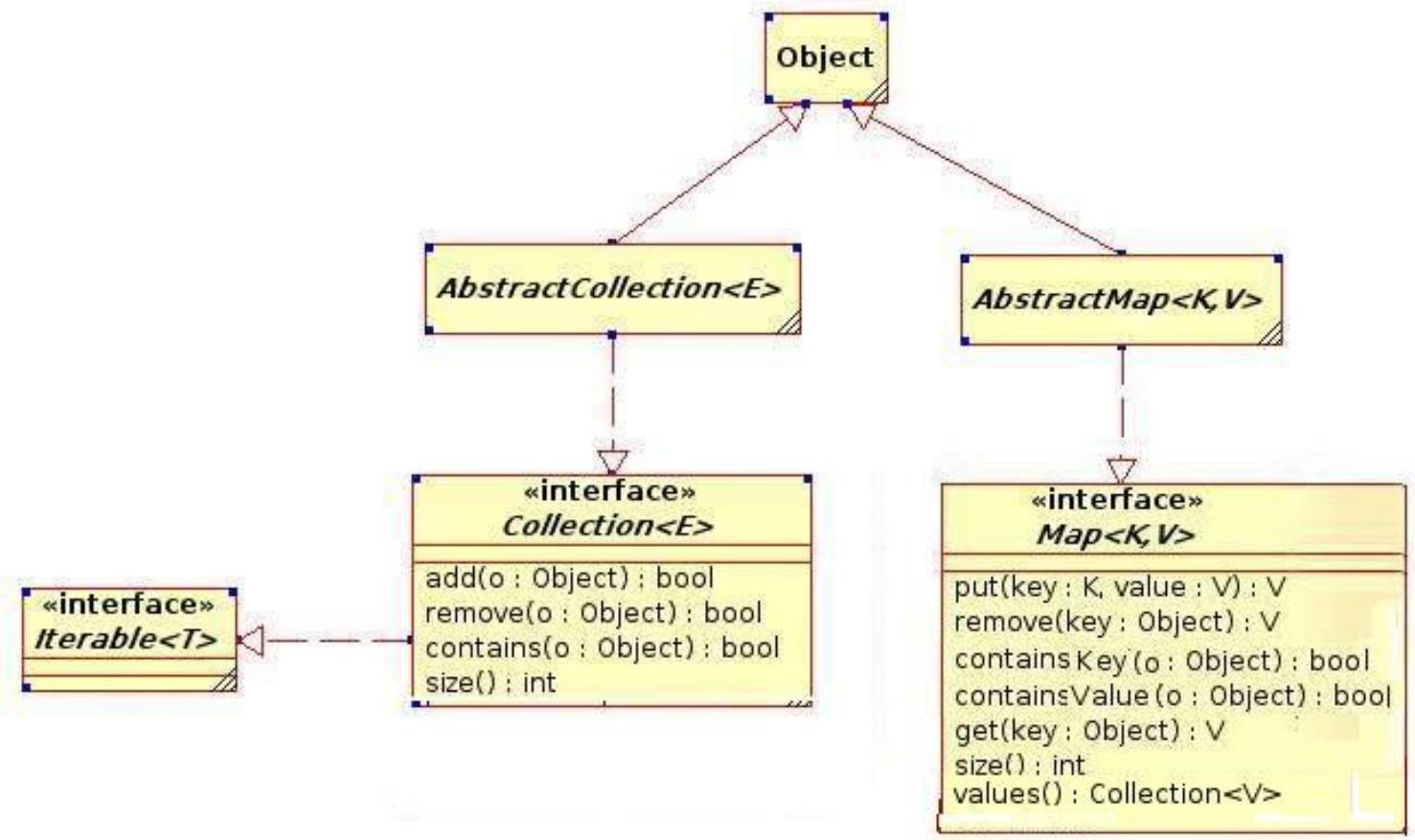
The enhanced `for` statement has the form:

EnhancedForStatement:
`for (` *VariableModifiers*$_{opt}$ *Type Identifier*`:` Expression`)` *Statement*

The *Expression* must either have type `Iterable` or else it must be of an array type (§10.1), or a compile-time error occurs.

Contexto java

Recorriendo



Iterable

Interface Iterable<T>

Method Summary

Iterator<T>	**iterator**() Returns an iterator over a set of elements of type T.

Method Detail

iterator

Iterator<T> **iterator**()

Returns an iterator over a set of elements of type T.

Returns:
an Iterator.

java.util

Interface Iterator<E>

Method Summary

boolean	**hasNext**() Returns true if the iteration has more elements.

Conceptos

Operaciones básicas

- Crear
- Adicionar un objeto a la colección
- Eliminar un objeto de la colección

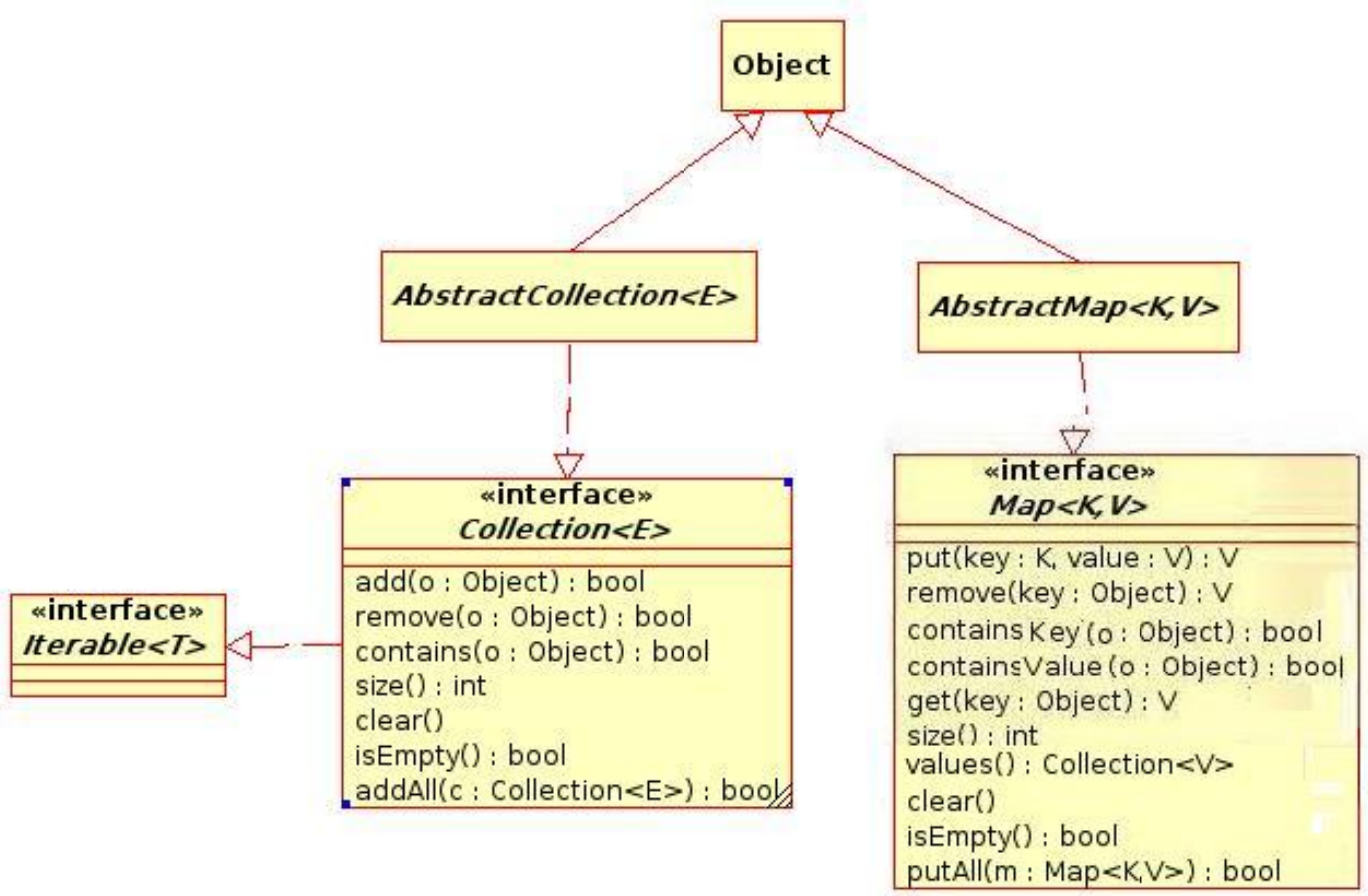
Operaciones-analizadoras

- ¿Cuántos elementos hay en la colección?
- ¿Qué elemento está en una posición de la colección?
-
- ¿Está un elemento específico en la colección?

Otras operaciones

Contexto java

Otras operaciones



Agenda

En general

```
import java.util.*;

public class ArrayListExample {
    public static void main(String[] args) {
        // Instantiate a collection.
        ArrayList<Student> students = new ArrayList<Student>();

        // Create a few Student objects.
        Student a = new Student();
        Student b = new Student();
        Student c = new Student();

        // Store references to all three Students in the collection.
        students.add(a);
        students.add(b);
        students.add(c);

        // ... and then iterate through them one by one,
        // printing each student's name.
        for (Student s : students) {
            System.out.println(s.getName());
        }
    }
}
```

En general

```
import java.util.*;

public class ArrayListExample {
    public static void main(String[] args) {
        // Instantiate a collection.
        ArrayList<Student> students = new ArrayList<Student>();

        // Create a few Student objects.
        Student a = new Student();
        Student b = new Student();
        Student c = new Student();

        // Store references to all three Students in the collection.
        students.add(a);
        students.add(b);
        students.add(c);

        // ... and then iterate through them one by one,
        // printing each student's name.
        for (Student s : students) {
            System.out.println(s.getName());
        }
    }
}
```

¿De qué otro tipo puede ser students sin cambiar código?

En general

```
import java.util.HashMap;

public class HashMapExample {
    public static void main(String[] args) {
        // Instantiate a HashMap with String as the key type and Student as
        // the value type.
        HashMap<String, Student> students = new HashMap<String, Student>();

        // Instantiate three Students; the constructor arguments are
        // used to initialize Student attributes idNo and name,
        // respectively, which are both declared to be Strings.
        Student s1 = new Student("12345-12", "Fred");
        Student s2 = new Student("98765-00", "Barney");
        Student s3 = new Student("71024-91", "Wilma");

        // Insert all three Students into the HashMap, using their idNo
        // as a key.
        students.put(s1.getIdNo(), s1);
        students.put(s2.getIdNo(), s2);
        students.put(s3.getIdNo(), s3);
```

En general

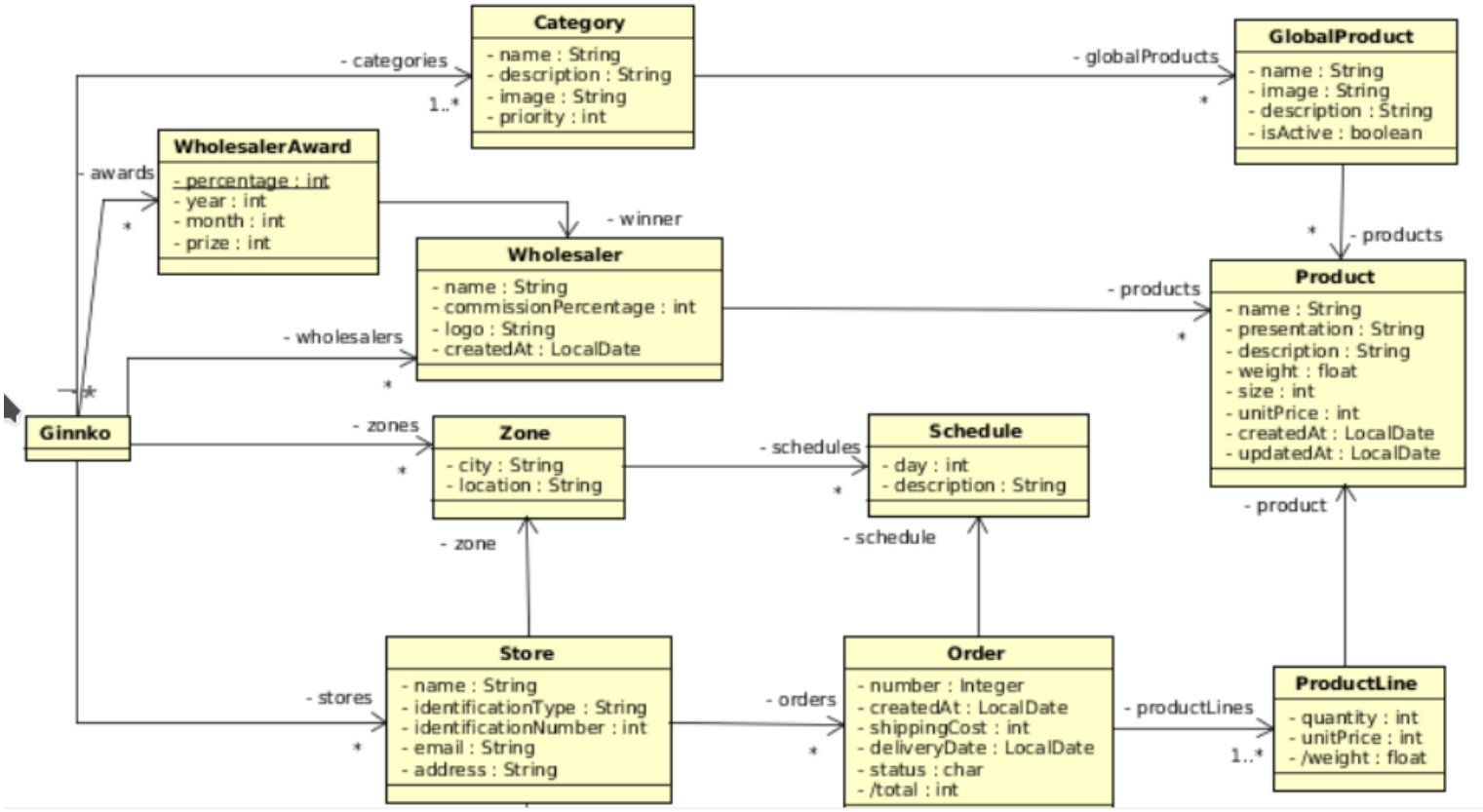
```
// Retrieve a Student based on a particular (valid) ID.
String id = "98765-00";
System.out.println("Let's try to retrieve a Student with ID = " + id);
Student x = students.get(id);

if (x != null) {
    System.out.println("Found!  Name = " + x.getName());
}
// ... whereas if the value returned was null, then we didn't find
// a match on the id that was passed in as an argument to get().
else {
    System.out.println("Invalid ID:  " + id);
}
```

En general

```
System.out.println();
System.out.println("Here are all of the students:");
System.out.println();

// Iterate through the HashMap to process all Students.
for (Student s : students.values()) {
    System.out.println("ID:   " + s.getIdNo());
    System.out.println("Name:   " + s.getName());
    System.out.println();
}
```

¿De qué otro tipo puede ser Students sin cambiar código?

Parcial



Diseñando

- Adicionar una distribuidora
- Consultar una tienda dado su nombre o su tipo y número de identificación
- Retornar el número de ordenes en cada estado

Agenda

Object

Constructor Summary
Object()

Method Summary	
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
int	hashCode() Returns a hash code value for the object.

Todos usan **equals**. Si es necesario se debe definir.

Las Hash usan **hashCode**

Comparable

java.lang

Interface Comparable<T>

Method Summary

int	**compareTo**(T o) Compares this object with the specified object for order.

Method Detail

compareTo

```
int compareTo(T o)
```

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Las claves del `Tree` deben implementar la interfaz `Comparable`

Agenda

Colecciones propias

Aproximaciones

1. Crear la clase desde cero
2. Extender una clase colección predefinida
3. Crear una clase que tenga como un atributo la colección predefinida

Laboratorios

Laboratorio 2

```java
import java.util.Stack;

/** Calculadora.java
 * Representa una calculadora de conjuntos
 * @author ESCUELA 2020-2
 */

public class CalConjuntos{

    private Stack<Conjunto> operandos;
    //Consultar en el API Java la clase Stack
```

Laboratorio 3

```java
public class Isla{
    public static final int MAXIMO = 500;
    private static Isla isla = null;

    public static Isla demeIsla() {
        if (isla==null){
            isla=new Isla();
        }
        return isla;
    }

    private static void nuevaIsla() {
        isla=new Isla();
    }

    public static void cambieIsla(Isla d) {
        isla=d;
    }

    private ArrayList<EnIsla> elementos;
    private int tesoroPosX;
    private int tesoroPosY;
    private boolean encontraronTesoro;
```

Colecciones propias

Opción dos

```
import java.util.ArrayList;

public class MyIntCollection extends ArrayList<Integer> {
   private int smallestInt;
   private int largestInt;
   private int total;


   public MyIntCollection() {
       super();
       total = 0;
    }
   public boolean add(int i) {
        if (this.isEmpty()) {
            smallestInt = i;
            largestInt = i;
        }
        else {
            if (i < smallestInt) smallestInt = i;
            if (i > largestInt) largestInt = i;
        }
        total = total + i;
        return super.add(i);
    }
```

Colecciones propias

Opción dos

```
    // Several new methods.

    public int getSmallestInt() {
        return smallestInt;
    }

    public int getLargestInt() {
        return largestInt;
    }

    public double getAverage() {
        // Note that we must cast ints to doubles to avoid
        // truncation when dividing.
        return ((double) total) / ((double) this.size());
    }
}
```

Colecciones propias

Opción dos

```
    // Several new methods.

    public int getSmallestInt() {
        return smallestInt;
    }

    public int getLargestInt() {
        return largestInt;
    }

    public double getAverage() {
        // Note that we must cast ints to doubles to avoid
        // truncation when dividing.
        return ((double) total) / ((double) this.size());
    }
}
```

¿Qué es bueno?¿Qué es malo?

Colecciones propias

Opción tres

```
import java.util.ArrayList;
public class MyIntCollection2 {
    // Instead, we're encapsulating a ArrayList inside of this class.
    ArrayList<Integer> numbers;
    private int smallestInt;
    private int largestInt;
    private int total;
    public MyIntCollection2() {
        numbers = new ArrayList<Integer>();
        total = 0;
    }
    public boolean add(int i) {
        if (numbers.isEmpty()) {
            smallestInt = i;
            largestInt = i:
        }
        else {
            if (i < smallestInt) smallestInt = i;
            if (i > largestInt) largestInt = i;
        }
        total = total + i;
        return numbers.add(i);
    }
```

Colecciones propias

Opción tres

```
    public int getSmallestInt() {
        return smallestInt;
    }

    public int getLargestInt() {
        return largestInt;
    }

    public double getAverage() {
        return ((double) total)/this.size();
    }

    // Since we don't INHERIT a size() method any longer, let's add one!
    public int size() {
        // DELEGATION!
        return numbers.size();
    }
}
```

Colecciones propias

Opción tres

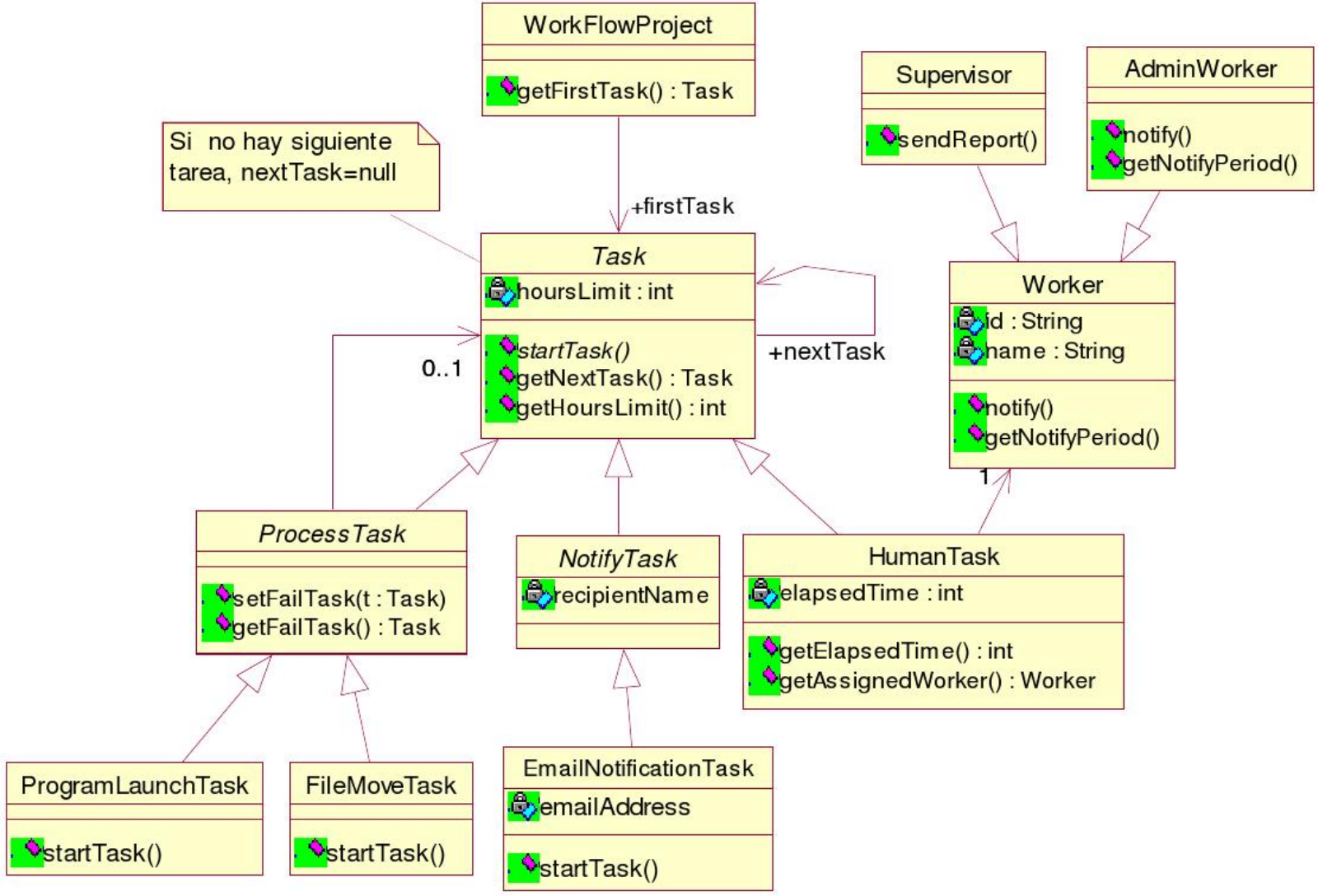
```
    public int getSmallestInt() {
        return smallestInt;
    }

    public int getLargestInt() {
        return largestInt;
    }

    public double getAverage() {
        return ((double) total)/this.size();
    }
    // Since we don't INHERIT a size() method any longer, let's add one!
    public int size() {
        // DELEGATION!
        return numbers.size();
    }
}
```

¿Qué es bueno?¿Qué es malo?

Agenda

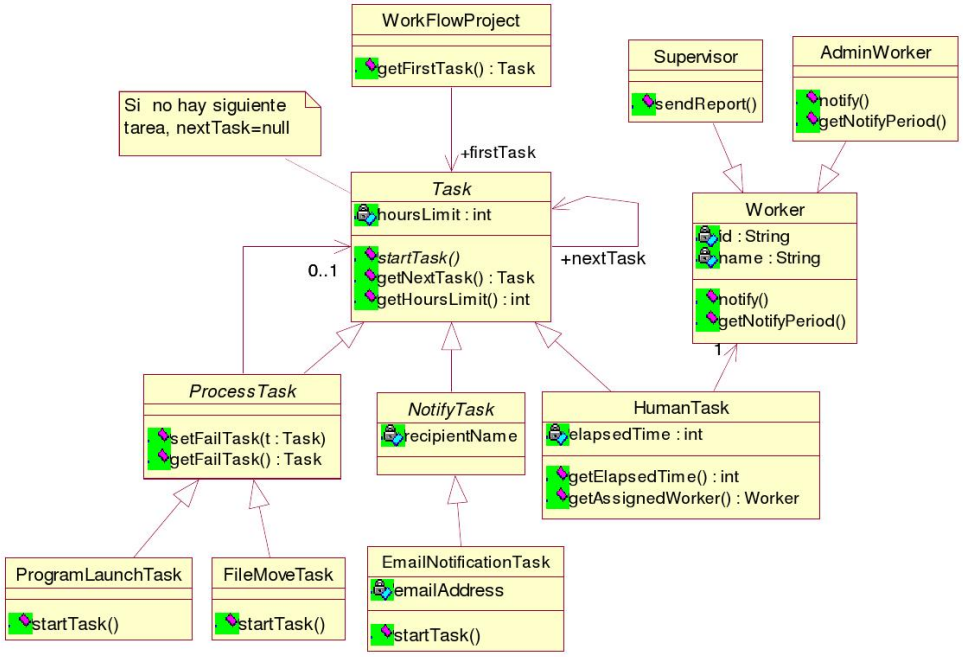
Ejemplos

Flujo de trabajo



Ejemplos

Flujo de trabajo

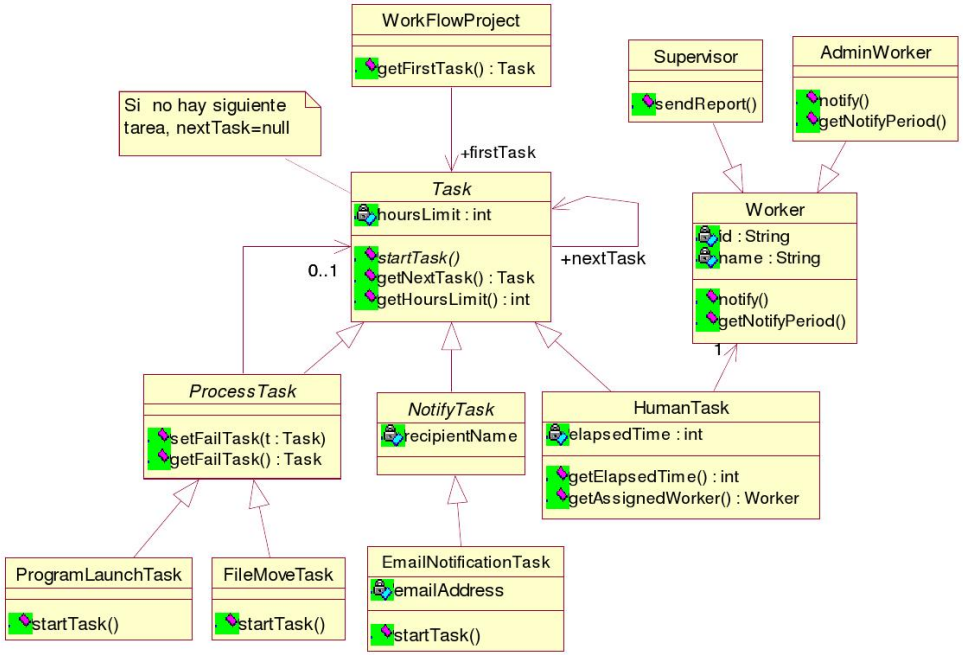


Analizando

1. ¿Qué colección tenemos?

Ejemplos

Flujo de trabajo

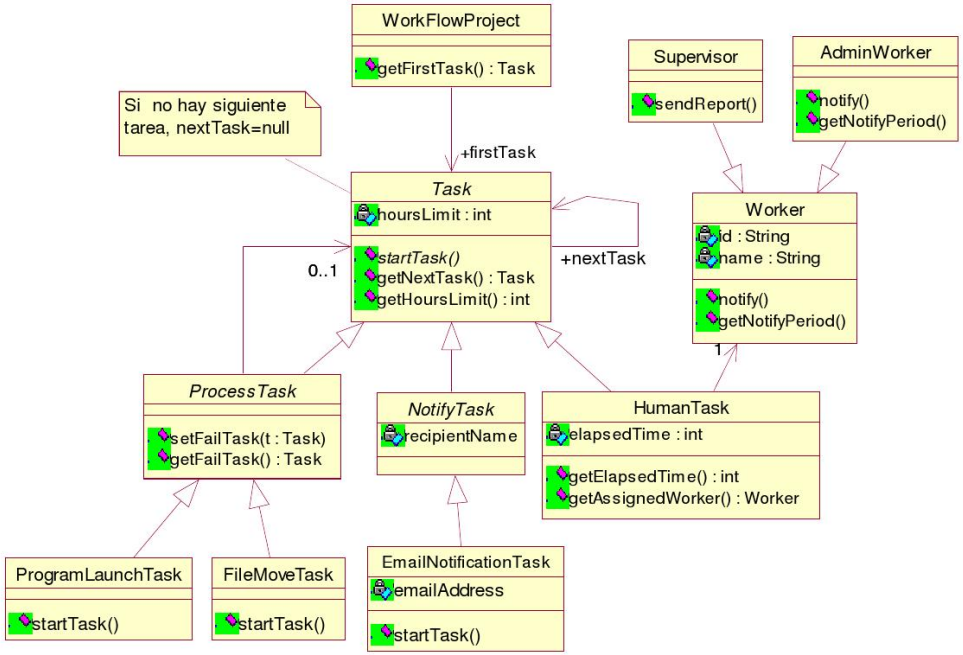


Analizando

1. ¿Qué colección tenemos?
2. ¿Qué alternativa se seleccionó?

Ejemplos

Flujo de trabajo



Analizando

1. ¿Qué colección tenemos?
2. ¿Qué alternativa se seleccionó?
3. ¿Cómo lo haríamos considerando otra alternativa?

Agenda

Colecciones propias

```
import java.util.ArrayList;
public class MyIntCollection2 {
    // Instead, we're encapsulating a ArrayList inside of this class.
    ArrayList<Integer> numbers;
    private int smallestInt;
    private int largestInt;
    private int total;
    public MyIntCollection2() {
        numbers = new ArrayList<Integer>();
        total = 0;
    }
    public boolean add(int i) {
        if (numbers.isEmpty()) {
            smallestInt = i;
            largestInt = i;
        }
        else {
            if (i < smallestInt) smallestInt = i;
            if (i > largestInt) largestInt = i;
        }
        total = total + i;
        return numbers.add(i);
    }
```

Colecciones propias

```
import java.util.ArrayList;
public class MyIntCollection2 {
    // Instead, we're encapsulating a ArrayList inside of this class.
    ArrayList<Integer> numbers;
    private int smallestInt;
    private int largestInt;
    private int total;
    public MyIntCollection2() {
        numbers = new ArrayList<Integer>();
        total = 0;
    }
    public boolean add(int i) {
        if (numbers.isEmpty()) {
            smallestInt = i;
            largestInt = i;
        }
        else {
            if (i < smallestInt) smallestInt = i;
            if (i > largestInt) largestInt = i;
        }
        total = total + i;
        return numbers.add(i);
    }
```

Colecciones propias

```
import java.util.ArrayList;
public class MyIntCollection2 {

    ArrayList<Integer> numbers;
    private int smallestInt;
    private int largestInt;

    public MyIntCollection2() {
        numbers = new ArrayList<Integer>();

    }
    public boolean add(int i) {
       if (numbers.isEmpty()) {
           smallestInt = i;
           largestInt = i;
       }
       else {
           if (i < smallestInt) smallestInt = i;
           if (i > largestInt) largestInt = i;
       }

      return numbers.add(i);
    }
```

Colecciones propias

Generica

```
import java.util.ArrayList;

public class MyCollection <E> {
        private ArrayList <E> collection;
        private E largest;
        private E smallest;

        public MyCollection (){
                collection=new ArrayList  <E>();
                largest=null;
                smallest=null;
        }

        public boolean  add (E element){

                if (collection.isEmpty()){
                        largest=element;
                        smallest=element;
                } else {
                        if (element.compareTo(largest)>0) largest=element;
                        if (element.compareTo(smallest) <0) smallest=element;
                }
                return collection.add(element);
        }

}
```

Colecciones propias

Generica

```
import java.util.ArrayList;

public class MyCollection <E> {
        private ArrayList <E> collection;
        private E largest;
        private E smallest;

        public MyCollection (){
                collection=new ArrayList  <E>();
                largest=null;
                smallest=null;
        }

        public boolean  add (E element){

                if (collection.isEmpty()){
                        largest=element;
                        smallest=element;
                } else {
                        if (element.compareTo(largest)>0) largest=element;
                        if (element.compareTo(smallest) <0) smallest=element;
                }
                return collection.add(element);
        }

}
```

Colecciones propias

Generica

```
import java.util.ArrayList;
import java.lang.Comparable;

public class MyCollection <E extends Comparable<E>>{
    private ArrayList <E> collection;
    private E largest;
    private E smallest;

    public MyCollection (){
        collection=new ArrayList  <E>();
        largest=null;
        smallest=null;
    }

    public boolean  add (E element){
        if ((collection.isEmpty()) || (element.compareTo(largest)>0)){
            largest=element;
        }
        if ((collection.isEmpty()) || (element.compareTo(largest)<0)){
           smallest=element;
        }
        return collection.add(element);
    }
```

Isla Coco

Requisitos

1. Necesitamos almacenar la información de los tesoros (x,y,$)
2. Existen muy pocos tesoros en la isla
3. Los queremos consultar por posición y ordenados por filas ((1,1)..(1,n),...(m,1),(m,n))

Isla Coco

Requisitos

1. Necesitamos almacenar la información de los tesoros (x,y,$)
2. Existen muy pocos tesoros en la isla
3. Los queremos consultar por posición y ordenados por filas ((1,1)..(1,n),...(m,1),(m,n))

Isla Coco

Requisitos

1. Necesitamos almacenar la información de los tesoros (x,y,$)
2. Existen muy pocos tesoros en la isla
3. Los queremos consultar por posición y ordenados por filas ((1,1)..(1,n),...(m,1),(m,n))
4. ¿Y si lo queremos ordenado también por columnas?

Otro orden

Constructor Summary
TreeMap() Constructs a new, empty map, sorted according to the keys' natural order.
TreeMap(Comparator<? super K> c) Constructs a new, empty map, sorted according to the given comparator.
TreeMap(Map<? extends K,? extends V> m) Constructs a new map containing the same mappings as the given map, sorted according to the keys' *natural order*.
TreeMap(SortedMap<K,? extends V> m) Constructs a new map containing the same mappings as the given SortedMap, sorted according to the same ordering.

Otro orden

Comparator

Method Summary	
int	**compare**(T o1, T o2) Compares its two arguments for order.
boolean	**equals**(Object obj) Indicates whether some other object is "equal to" this Comparator.