Programación Orientada a Objetos

Colecciones

CEIS

2023-2

Agenda

Agenda

Conceptos

En general

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

¿Qué colecciones conocen?

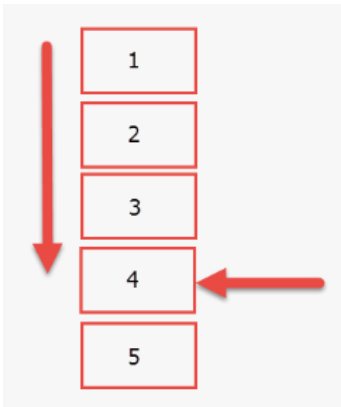
- AYPR - AYED
- LCAT - MATD

¿Cómo las categorizamos?

Conceptos

Operaciones-básicas

Operaciones-analizadoras



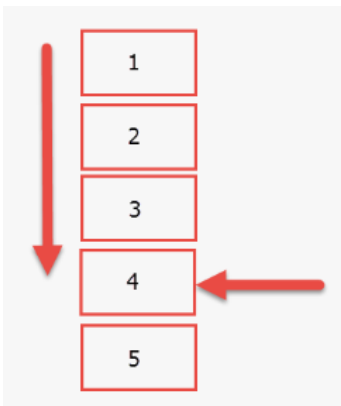
Conceptos

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?
- ¿ Qué elemento de la colección tiene una clave?

Conceptos

En POOB

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

¿Qué colecciones hemos manejado?

Laboratorios

Laboratorio 2

```
public class NPTensor{

    private HashMap<String, Tensor> variables;

    public NPTensor(){
    }
```

¿Qué contienen? ¿Qué permiten?

Laboratorios

Laboratorio 3

```
/*No olviden adicionar la documentacion*/
public class Colony{
    static private int LENGTH=30;
    private Entity[][] colony;
»
```

¿Qué contienen? ¿Qué permiten?

Laboratorios

Laboratorio 4

```
public class CostumeShop{
    private LinkedList<Costume> costumes;
    private HashMap<String,Basic> basics;
```

¿Qué contienen? ¿Qué permiten?

Agenda

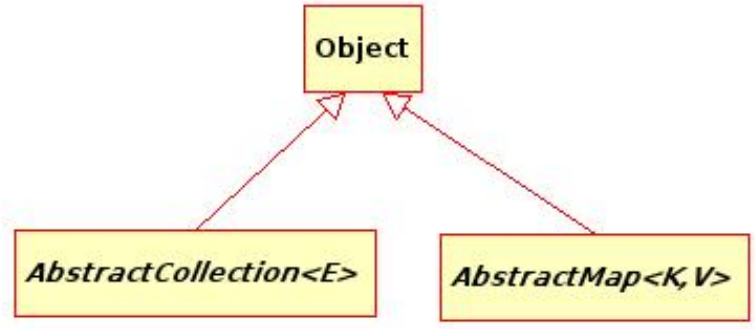
Conceptos

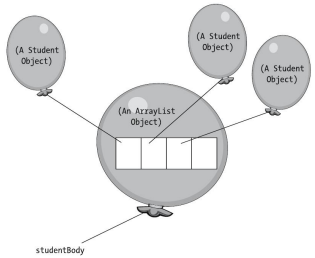
Tipos básicos

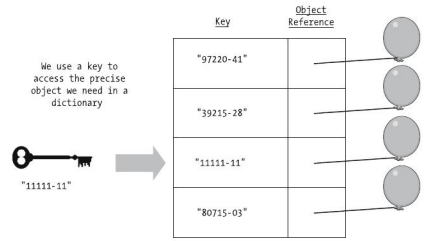
1. Colecciones simples
   - **Secuencias**
     Pueden existir elementos repetidos.

     Los distingue la posición.
   - **Conjuntos**
     No pueden existir elemento repetidos
2. Colecciones con clave
   - **Diccionarios**
     Almacena pares (clave-valor).
     No pueden existir claves repetidas

Dos clases abstractas

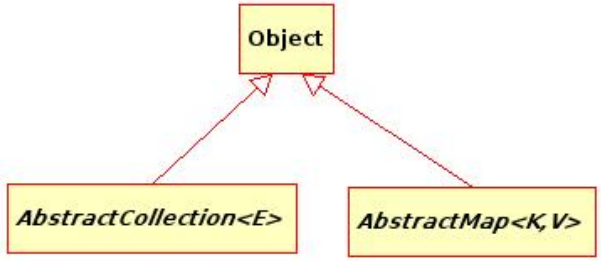




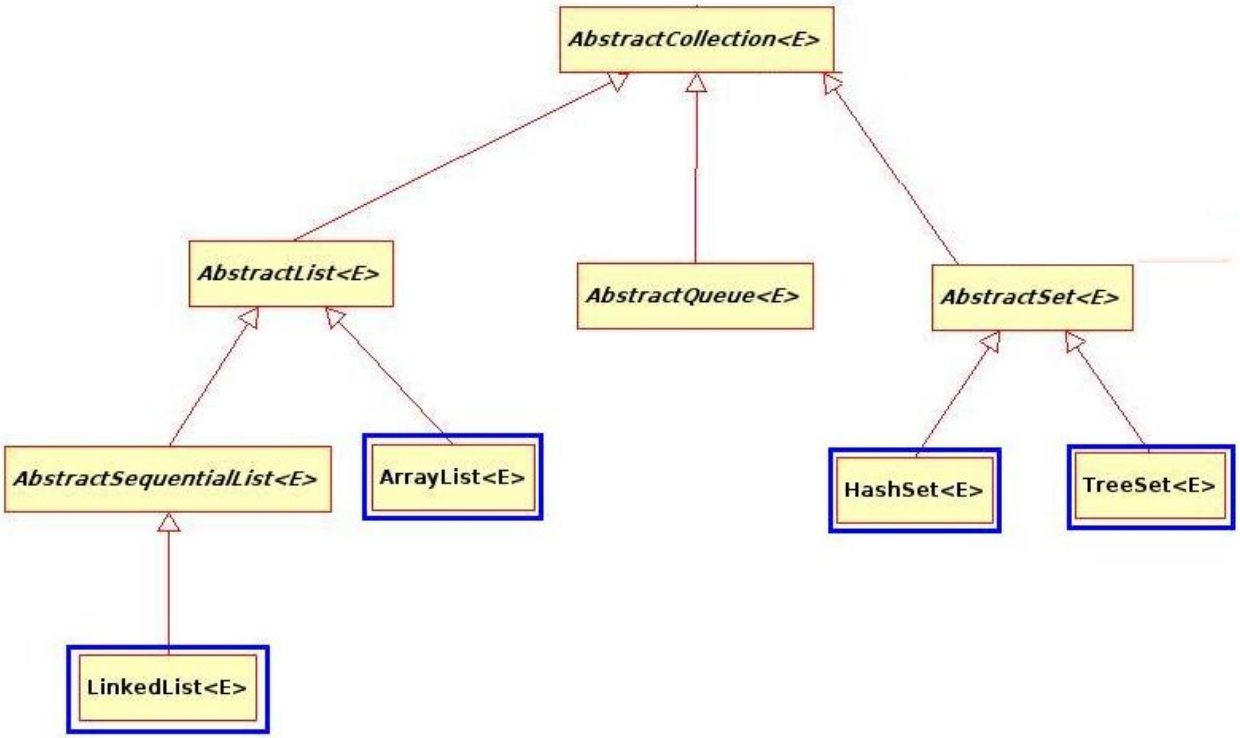


Contexto java

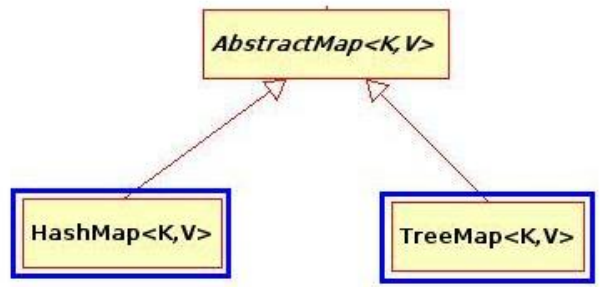
Dos clases abstractas



Sin clave



Con clave



En java

Tipos

1. Colecciones simples
   - Secuencias : `List`
     `ArrayList`,`LinkedList`
   - Conjuntos : `Set`
     `HashSet`, `TreeSet`
2. Colecciones con clave
   - Diccionarios : `Map`
     `HashMap`,`TreeMap`

En java

Tipos

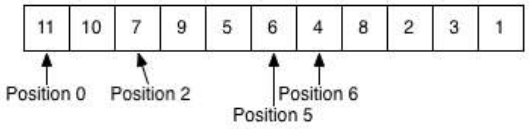
1. Colecciones simples
   - Secuencias : `List`
     `ArrayList`,`LinkedList`
   - Conjuntos : `Set`
     `HashSet`, `TreeSet`
2. Colecciones con clave
   - Diccionarios : `Map`
     `HashMap`,`TreeMap`

Representación

1. Arreglos : `Array`
   `ArrayList`
2. Listas enlazadas : `Linked`
   `LinkedList`
3. Tablas Hash: `Hash`
   `HashSet`, `HashMap`
4. Árboles: `Tree`
   `TreeSet`, `TreeMap`

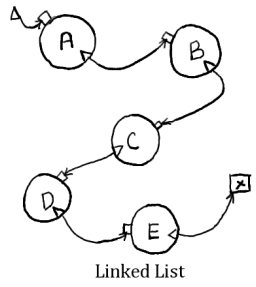
Representación

Array



Arreglo que puede cambiar de tamaño.

List



Listas enlazadas.

Hash

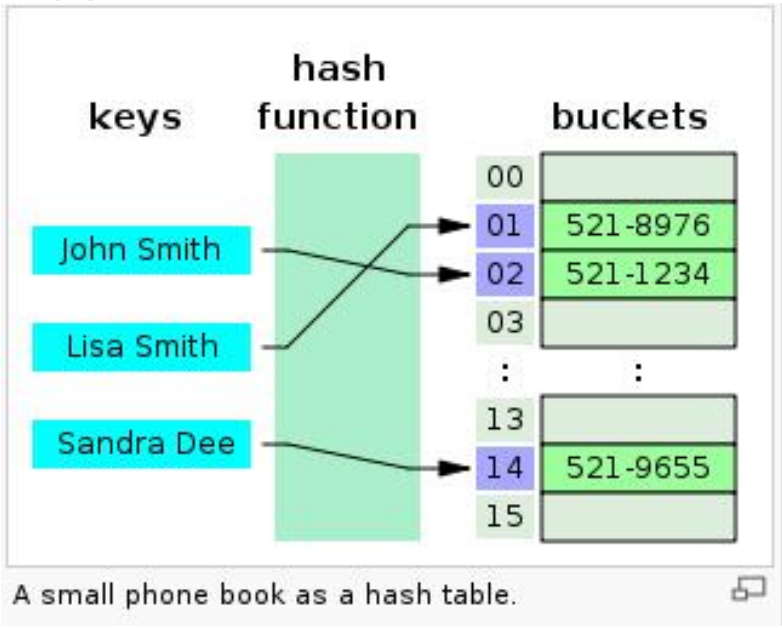
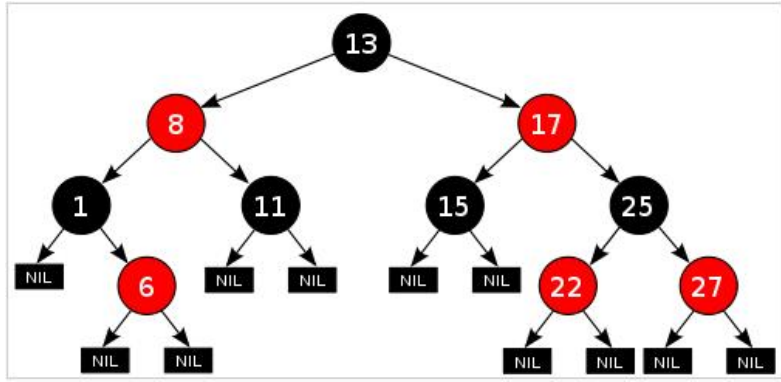


Tabla de hashing.

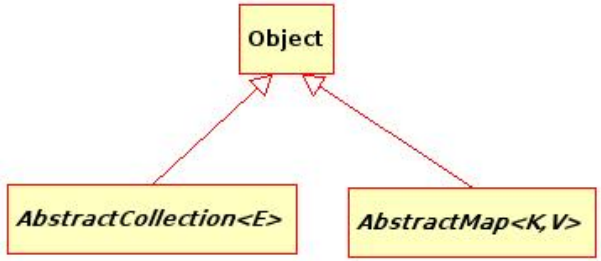
Tree



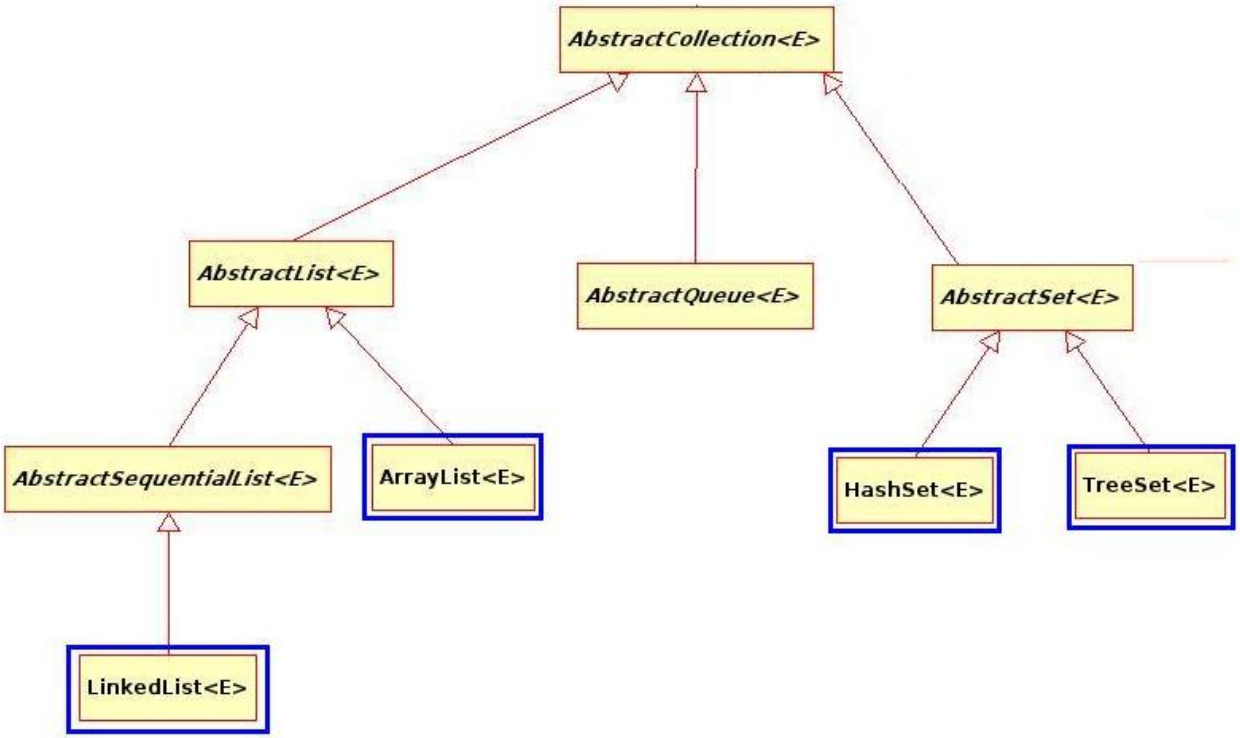
Árbol rojo-negro (ordenado - balanceado).

Contexto java

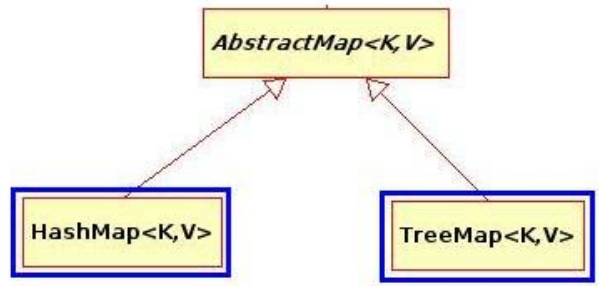
Dos objetos base



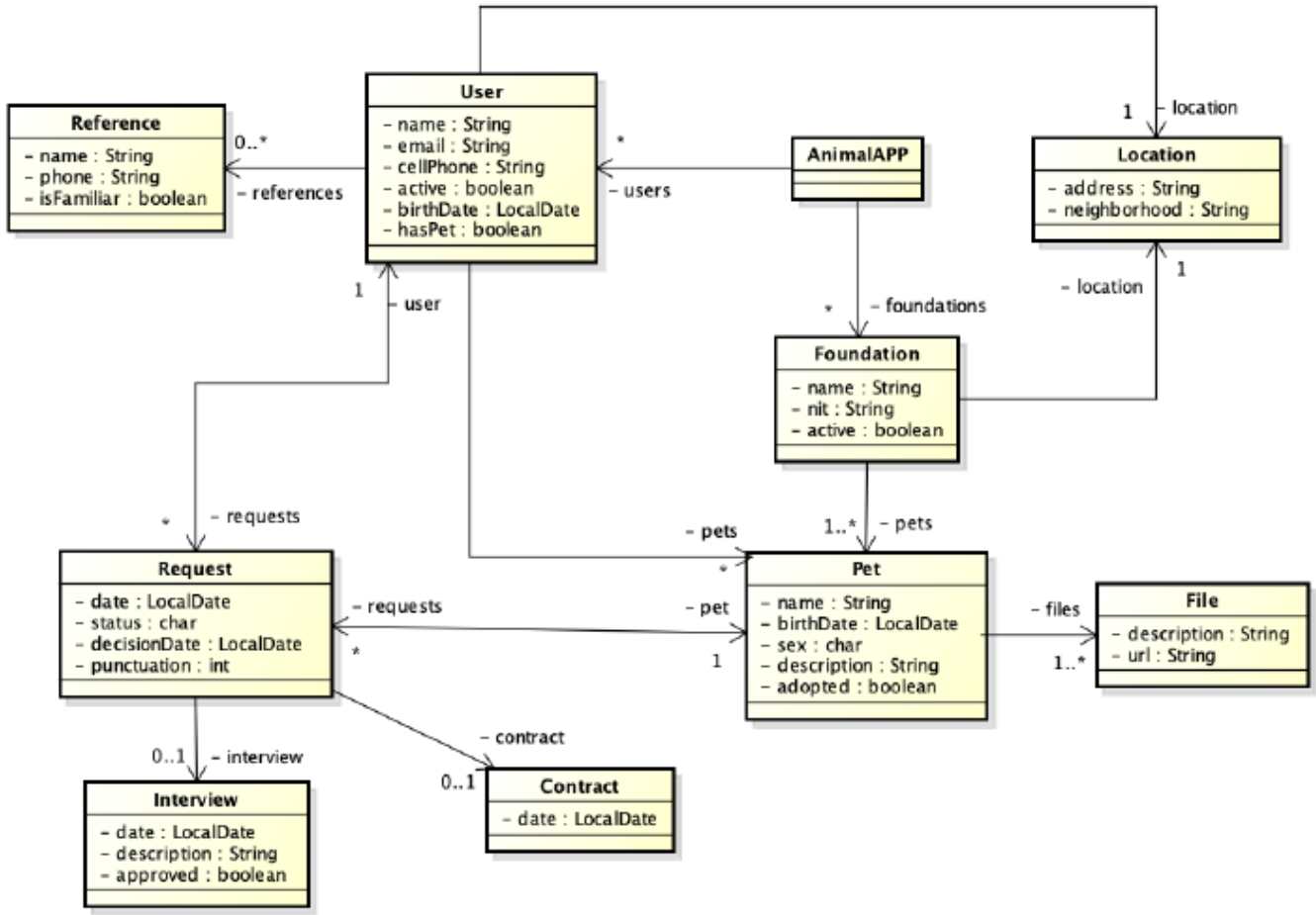
Sin clave



Con clave



Parcial



Diseñando

- Los usuarios los queremos consultar por correo (único), por telefono (único) y por nombre (no único). Los informes de los nombres los queremos ordenados alfabéticamente
- Las solicitudes las queremos mantener en el orden de realización
- Las macotas los queremos consultar rápidamente dado su nombre y fecha de nacimiento (no único)

Agenda

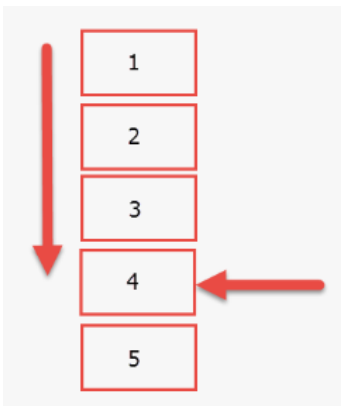
Conceptos

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?



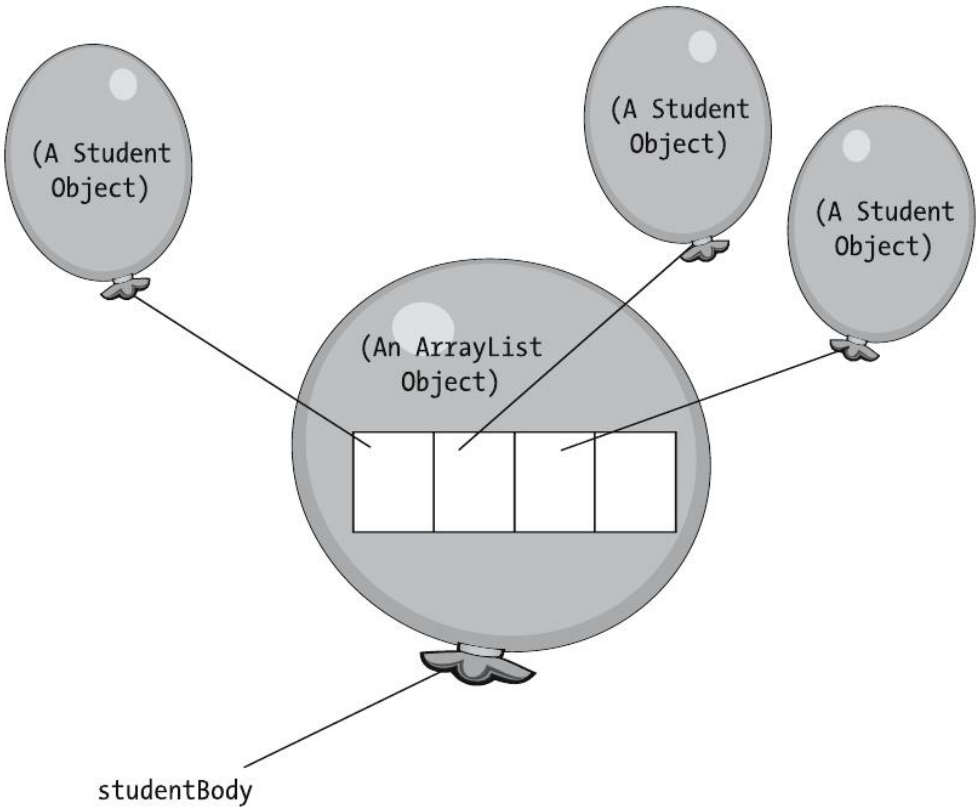
- ¿Está un elemento en la colección?
- ¿ Qué elemento de la colección tiene una clave?

Operadores: creación

Creando

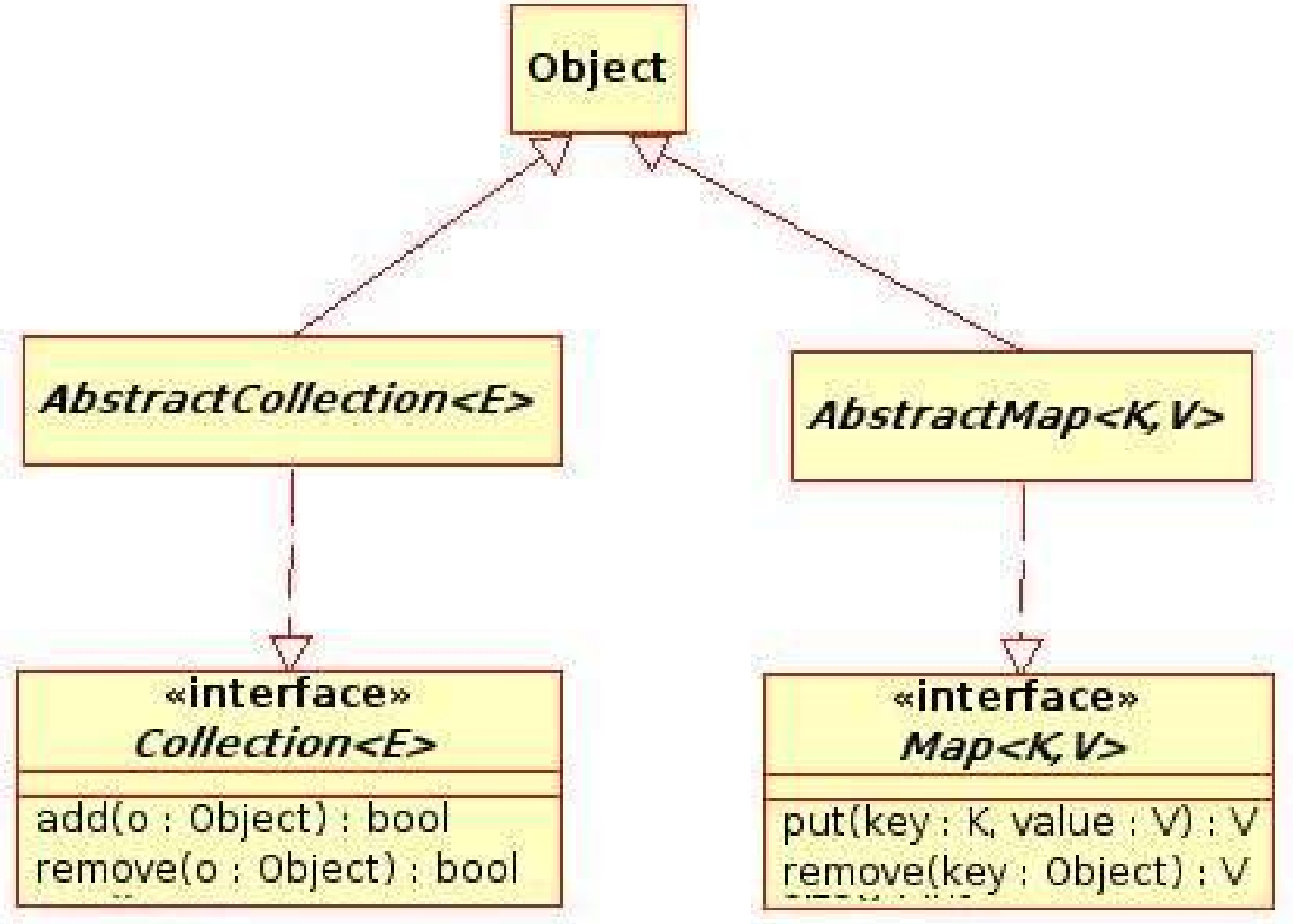
```
ArrayList<Student> studentBody ;  // ArrayList is one of Java's predefined collec
studentBody  = new ArrayList<Student>();
```

En Uso



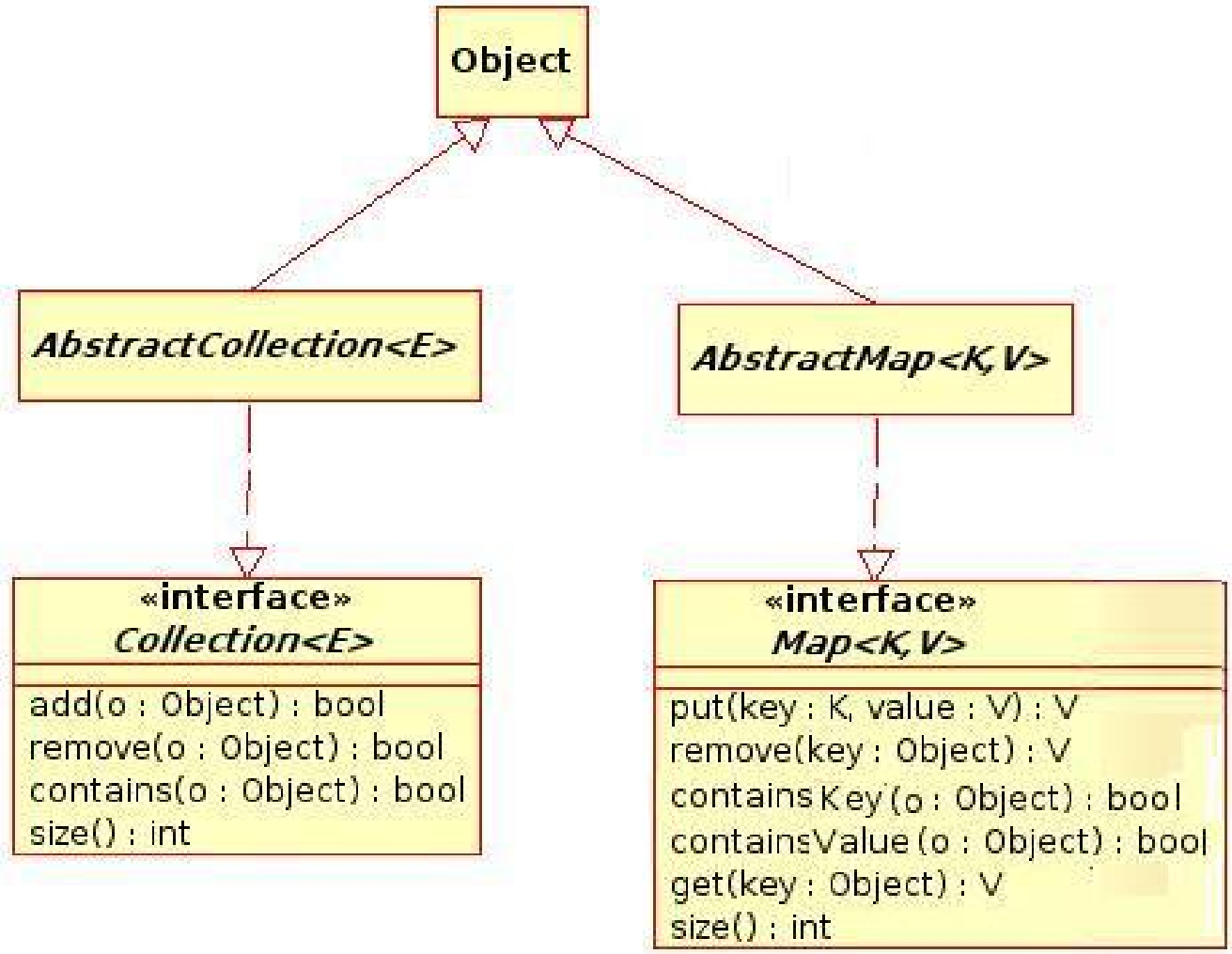
Contexto java

Operaciones básicas : adicionar y eliminar



Contexto java

Operaciones básicas



equals

Contexto java

Recorriendo

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

Especificación

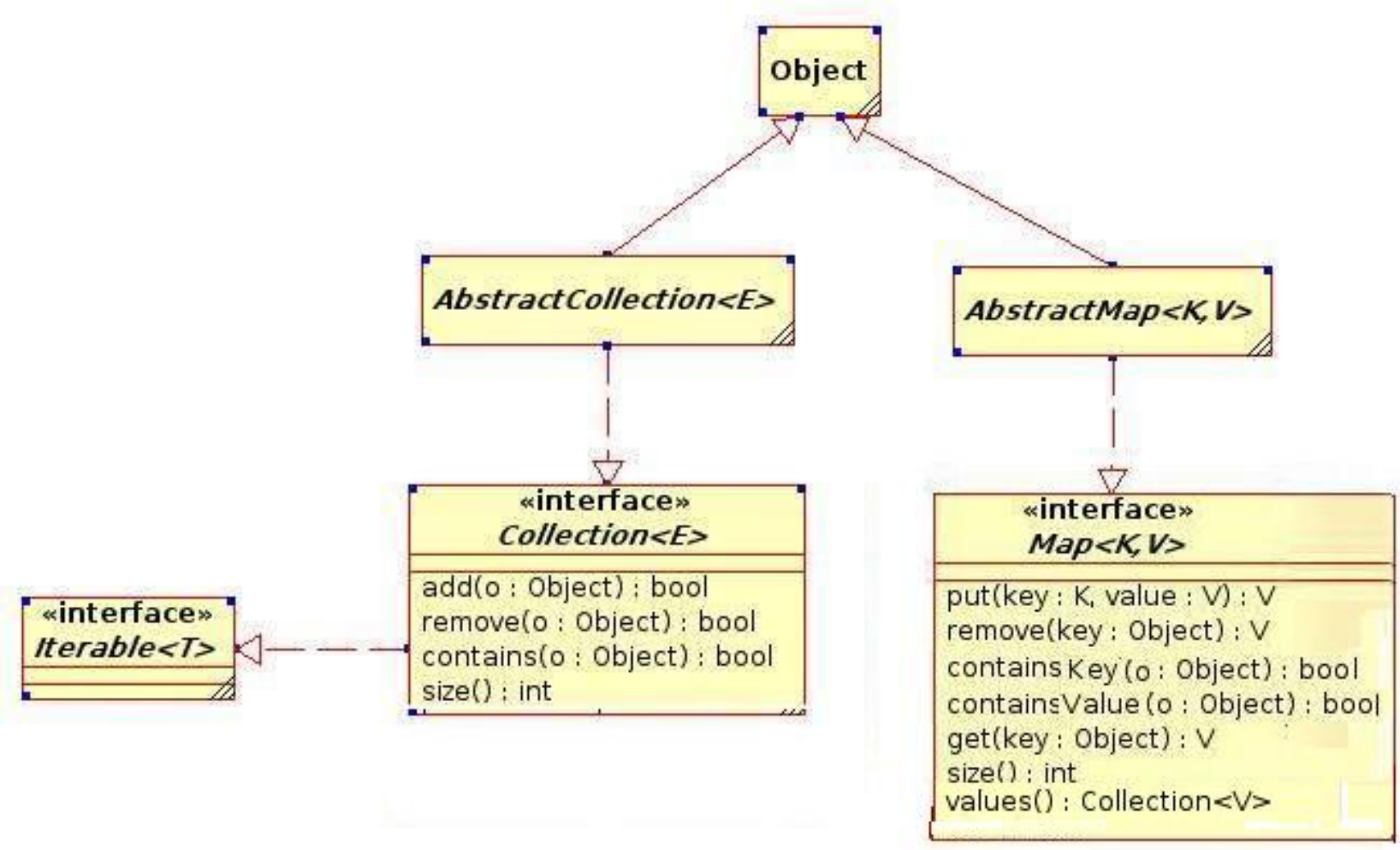
The enhanced `for` statement has the form:

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

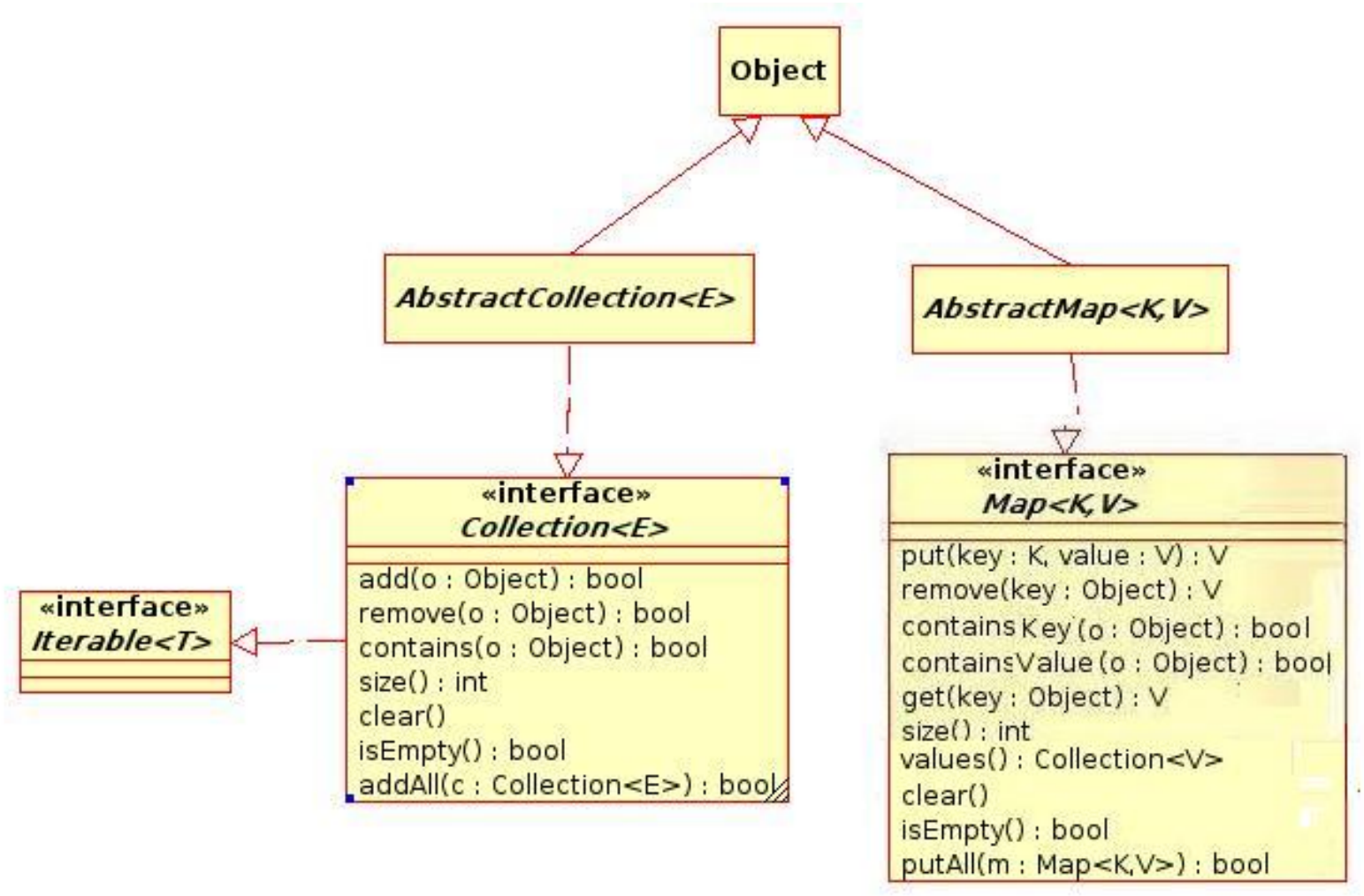
Interface Iterator<E>

Method Summary

boolean	**hasNext**() Returns `true` if the iteration has more elements.
E	**next**() Returns the next element in the iteration.

Contexto java

Otras operaciones



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

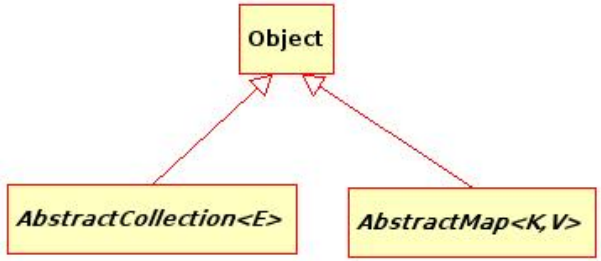
```
System.out.println();
System.out.println("Here are all of the students:");
System.out.println();

// 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();
}
```

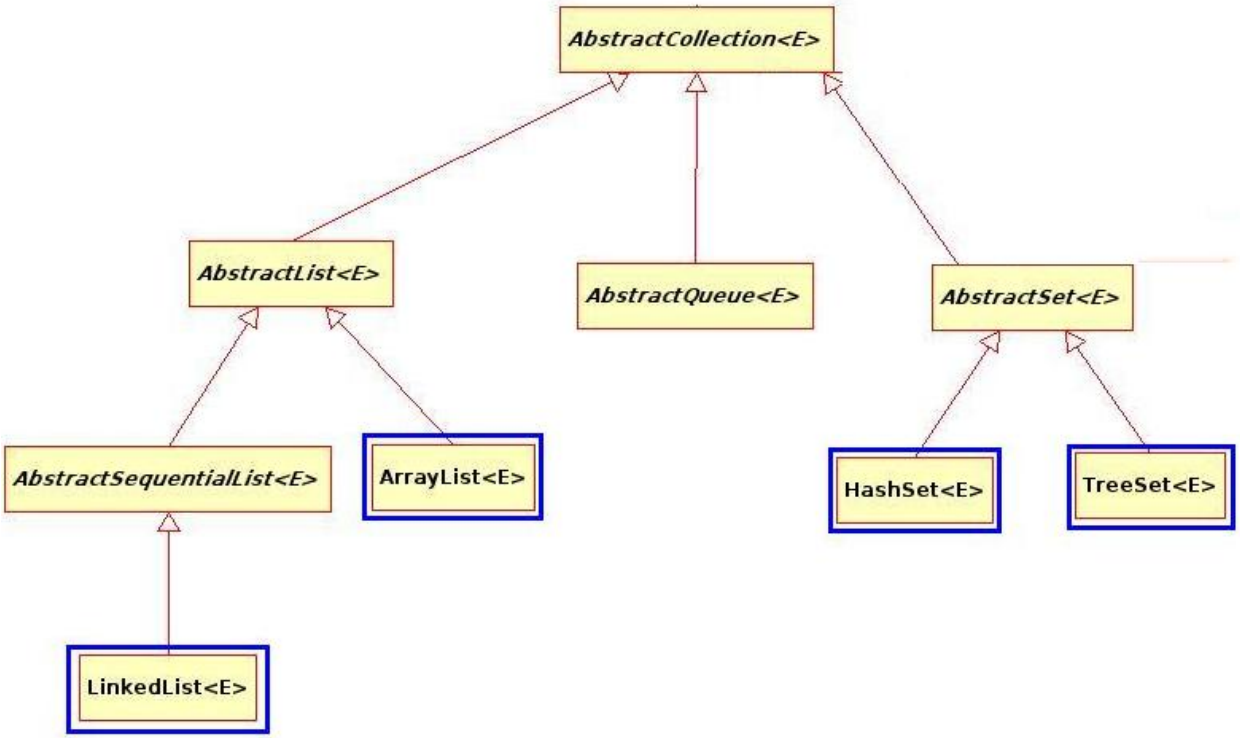
¿De qué otro tipo puede ser Students sin cambiar código?

Contexto java

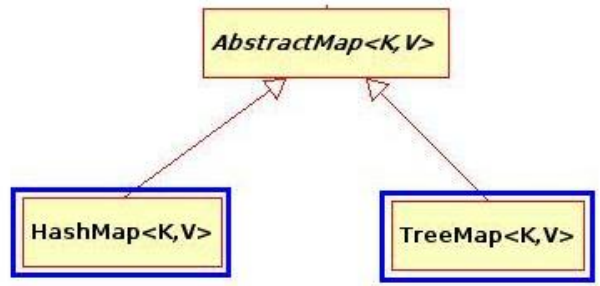
Dos objetos base



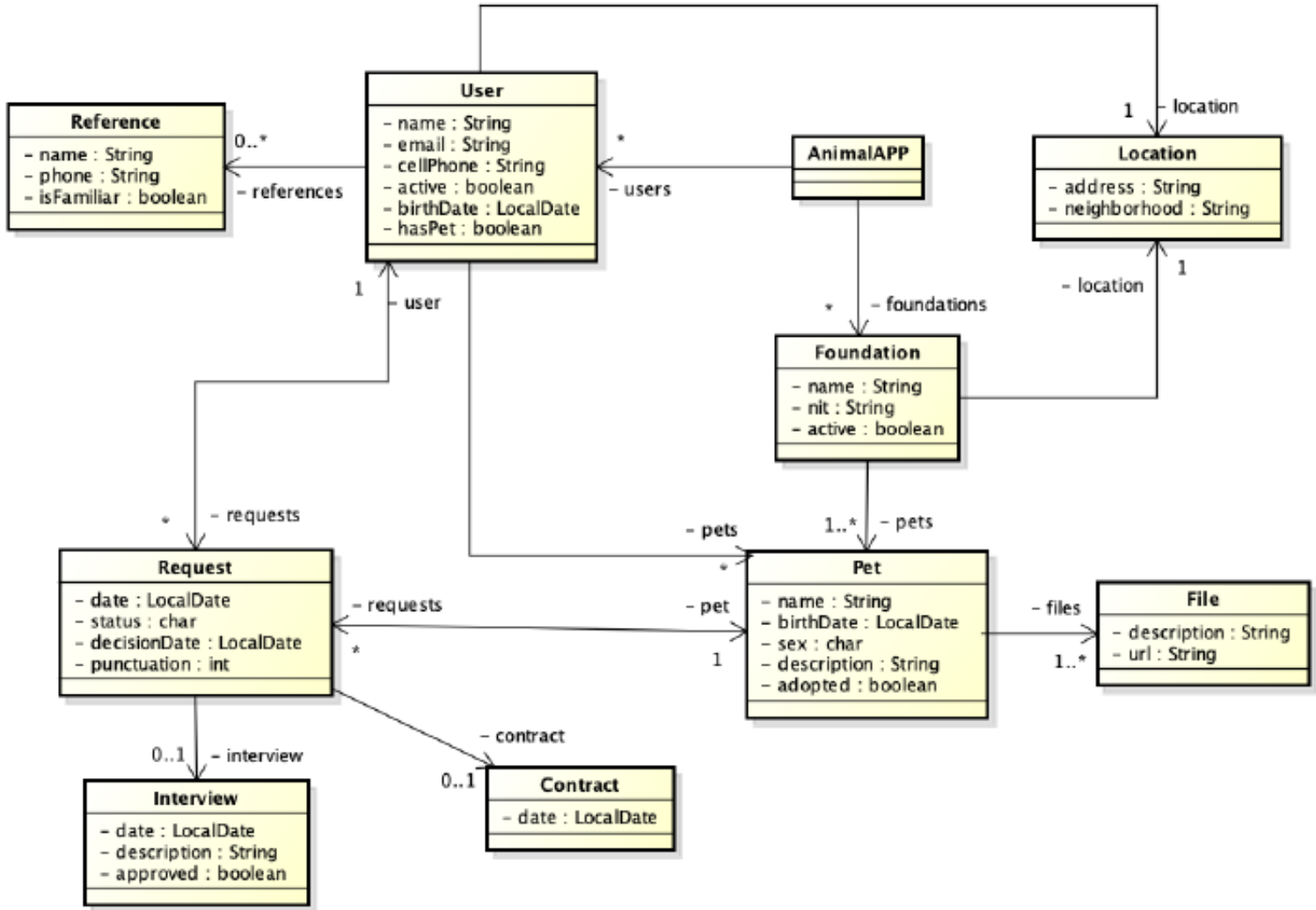
Sin clave



Con clave



Parcial



Diseñando

- Adicionar una persona dado nombre, correo, nacimiento y telefono (se supone nisn mascota y activo)
- Consultar una solicitud dado el correo de la persona y la fecha
- Consultar las mascotas de una fundación dado su nombre (ordenadas por fecha de nacimiento)

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

Alternativas: Laboratorio 4

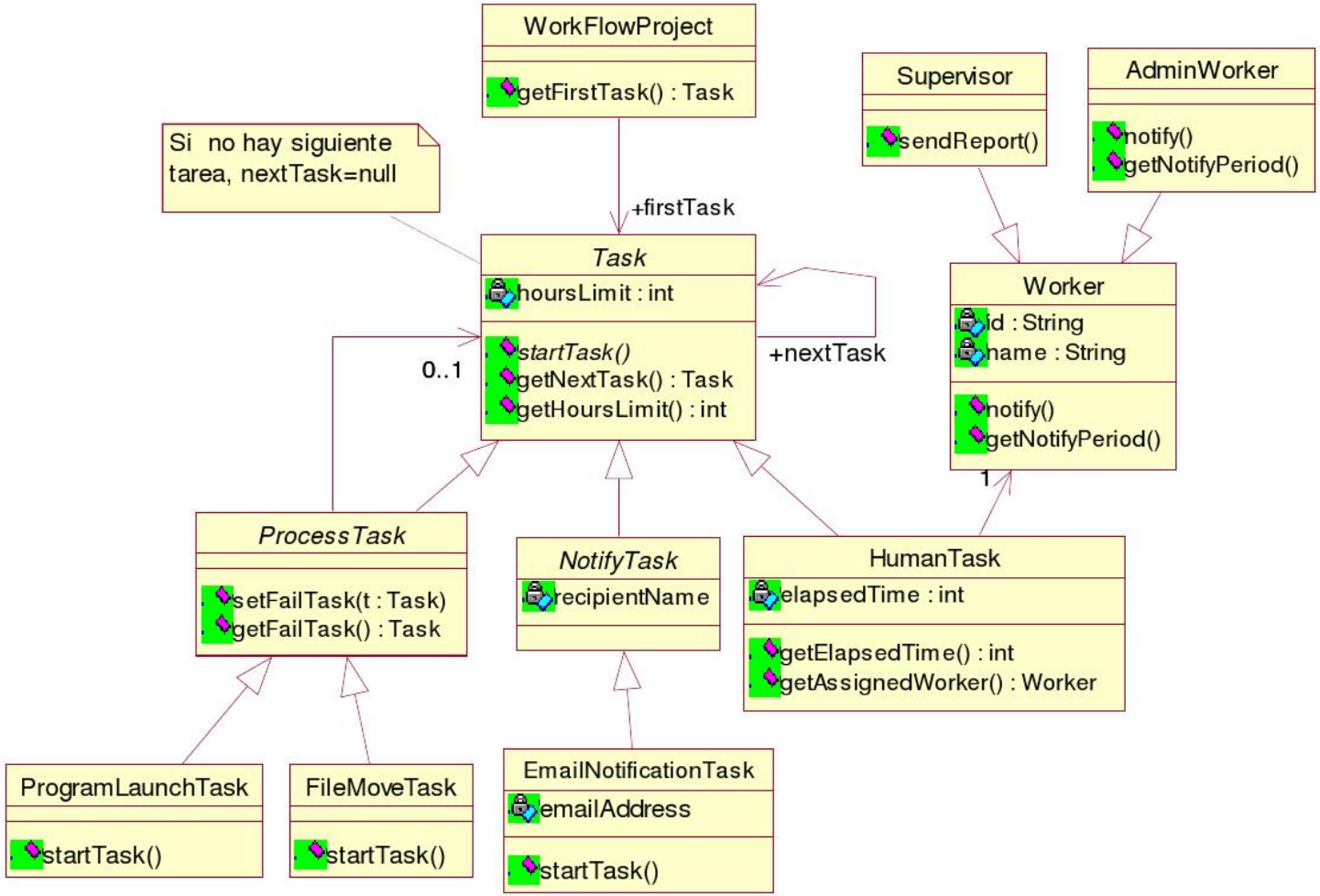
```
public class IEMOIS{
    private LinkedList<Program> programs;
    private HashMap<String,Course> courses;

public class CompoundActivity extends Activity{
    private ArrayList<Activity> activities;
```

Analizando

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

Alternativas. Flujo de Trabajo.



Analizando

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

Alternativas. My Collection

```
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);
    }

   // 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());
   }
}
```

Analizando

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

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

Genérica

```
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);
        }

}
```

¿Comparable?

Colecciones propias

Generica

```
import java.util.ArrayList;

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

}
```