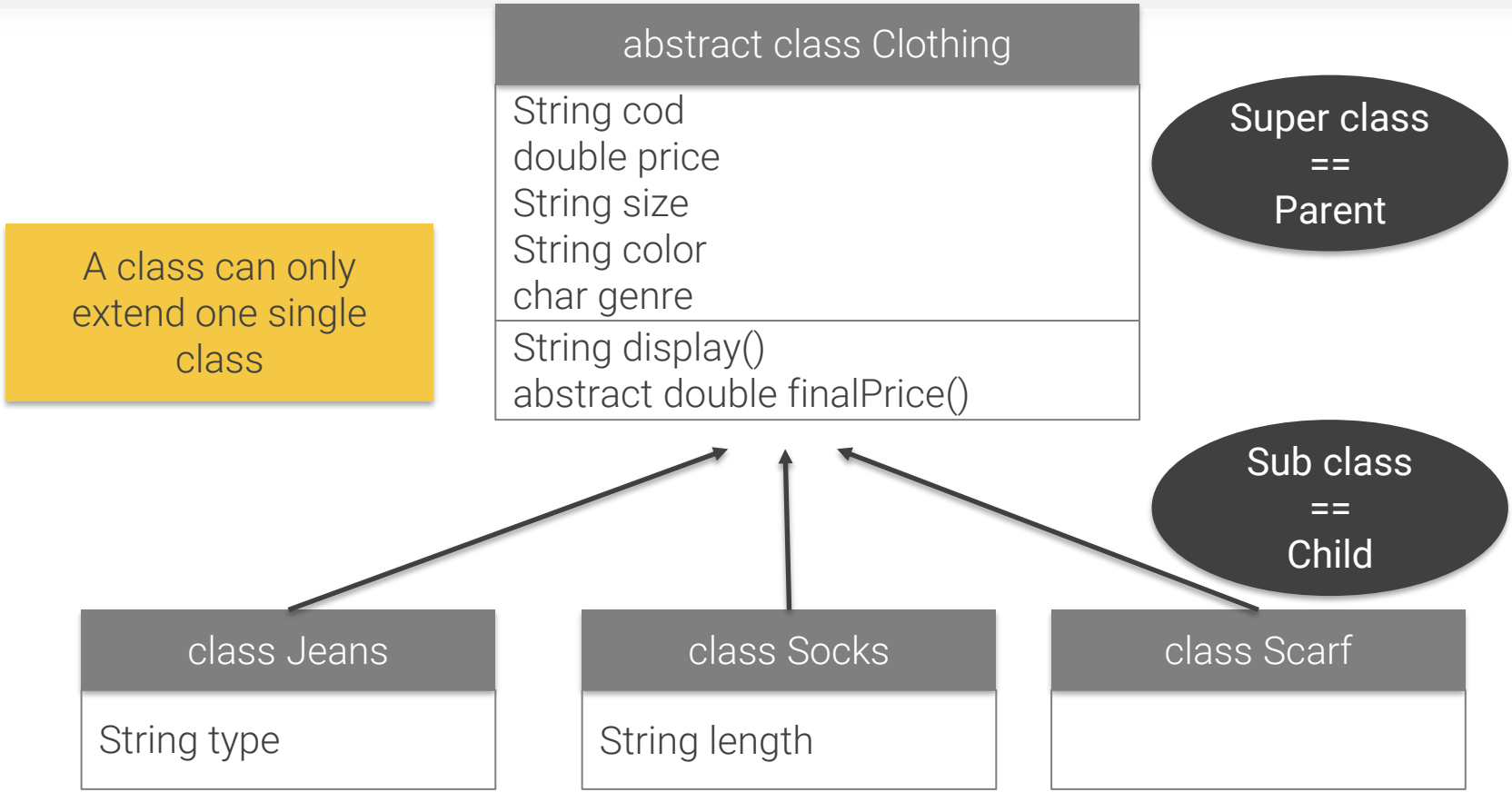
Multiple inheritance: Interfaces

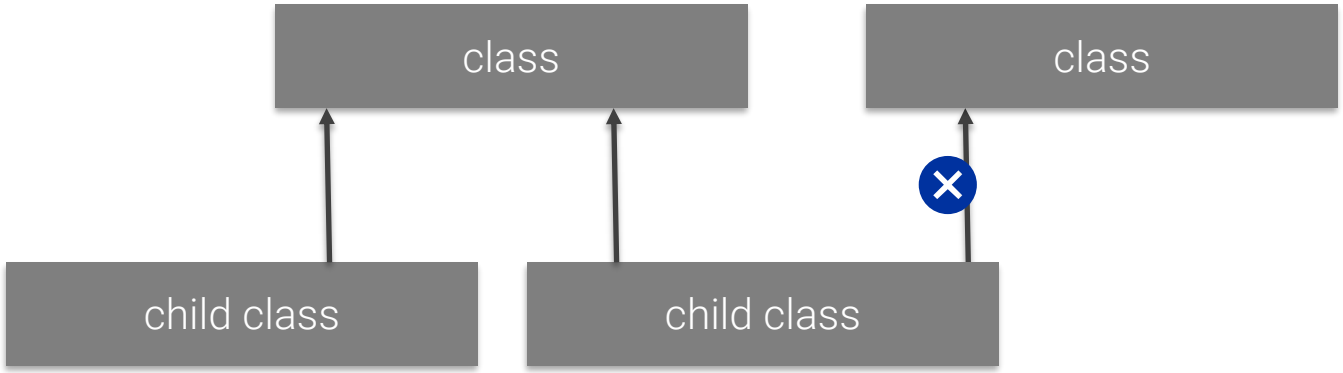
1. Multiple inheritance
2. Interfaces
3. Creating an interface in IntelliJ
4. Java predefined interfaces

Multiple inheritance

1. Multiple inheritance



1. Multiple inheritance



A CLASS CAN NOT EXTEND MORE THAN ONE CLASS

Why?

Because multiple inheritance could cause ambiguity if the parents (super classes) had similar methods.

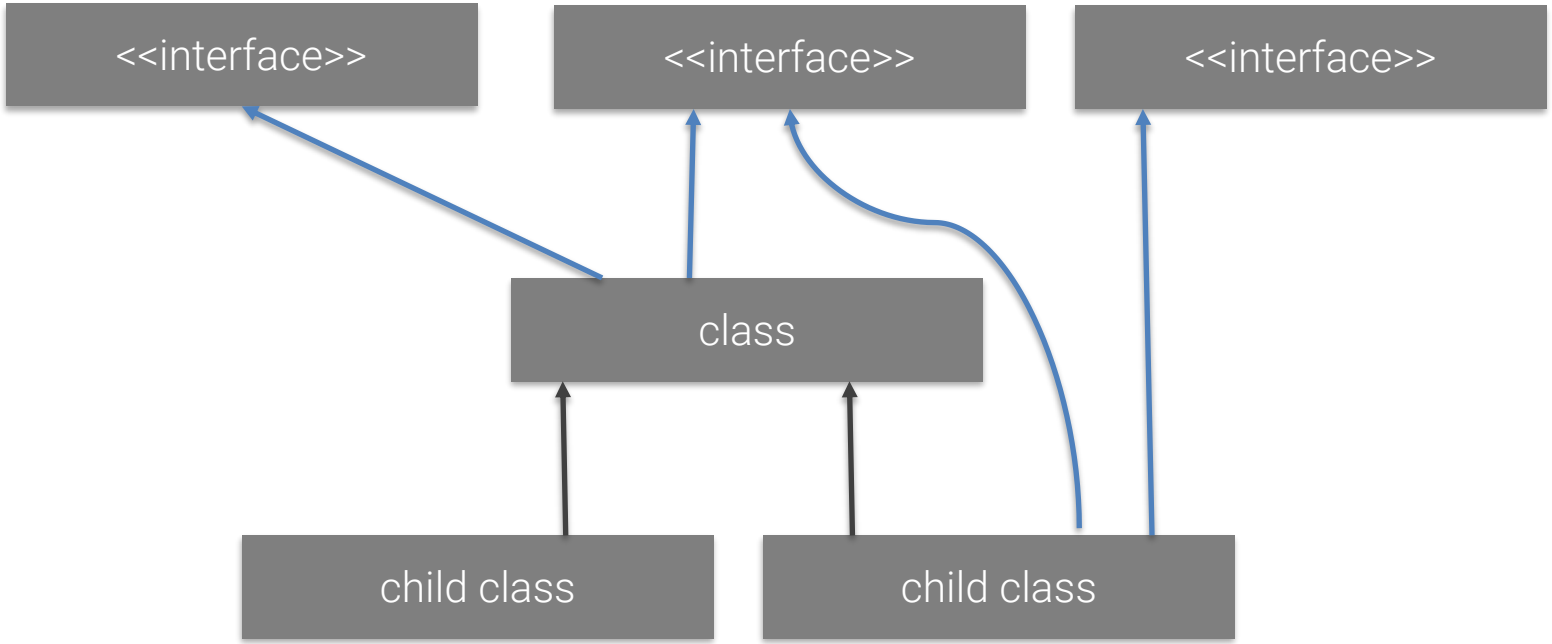
If you'd like to know more about: The Diamond Problem.

1. Multiple inheritance

A CLASS CAN NOT EXTEND MORE THAN ONE CLASS

Java solution to the multiple inheritance problem: **Interfaces**

1. Multiple inheritance



Interfaces

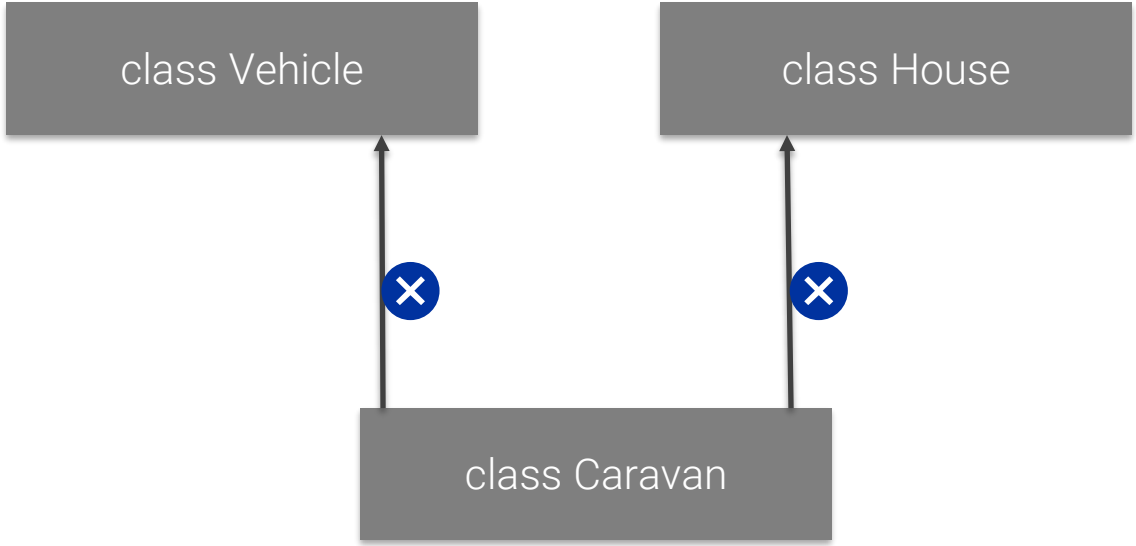
class Caravan

VEHICLE

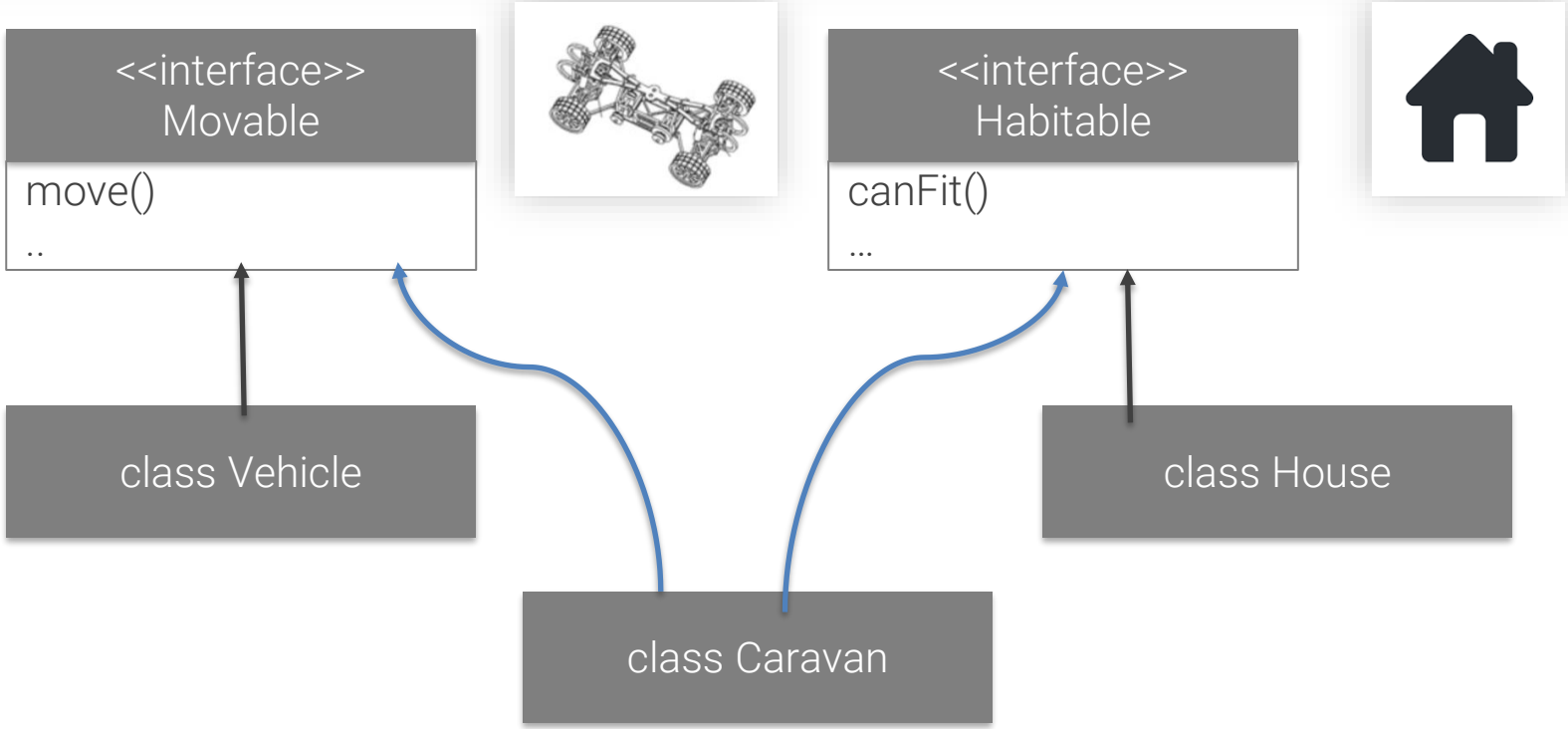
HOUSE

Fuente: www.pixabay.com

2. Interfaces



2. Interfaces



2. Interfaces

```
public interface Movable{

  void move(int distance);
  boolean canMove();

}
```

public and abstract methods

Only list method signatures

Implementation is responsability of the class that will implement the interface

Movable.java

2. Interfaces

```
public interface Habitable{

  boolean canFit(int inhabitants);

}
```

Habitable.java

2. Interfaces

```
public class Caravan implements Movable, Habitable{

}
```

Caravan.java

2. Interfaces

```
public class Caravan implements Movable, Habitable{

  void move(int distance) { ... }
  boolean canMove() { ... }
  boolean canFit(int inhabitants) { ... }

}
```

Implementation is responsability of the class that will implement the interface

Caravan.java

2. Interfaces

- Interfaces defines what a class should do but not how to do it.

2. Interfaces

- Interfaces defines what a class should do but not how to do it.
- You cannot create an instance (an object) from an interface. An interface's purpose is to be implemented by one or more classes

2. Interfaces

- Interfaces defines what a class should do but not how to do it.
- You cannot create an instance (an object) from an interface. An interface's purpose is to be implemented by one or more classes
- It's not reducing code repetition. Interfaces helps you about enforcing a good design

Creating an interface in IntelliJ

3. Creating an interface in IntelliJ

Hello World [~/IdeaProjects/Hello World] - .../src/com/company/Main.java [Hello World]

Hello World > src > com > company > Main

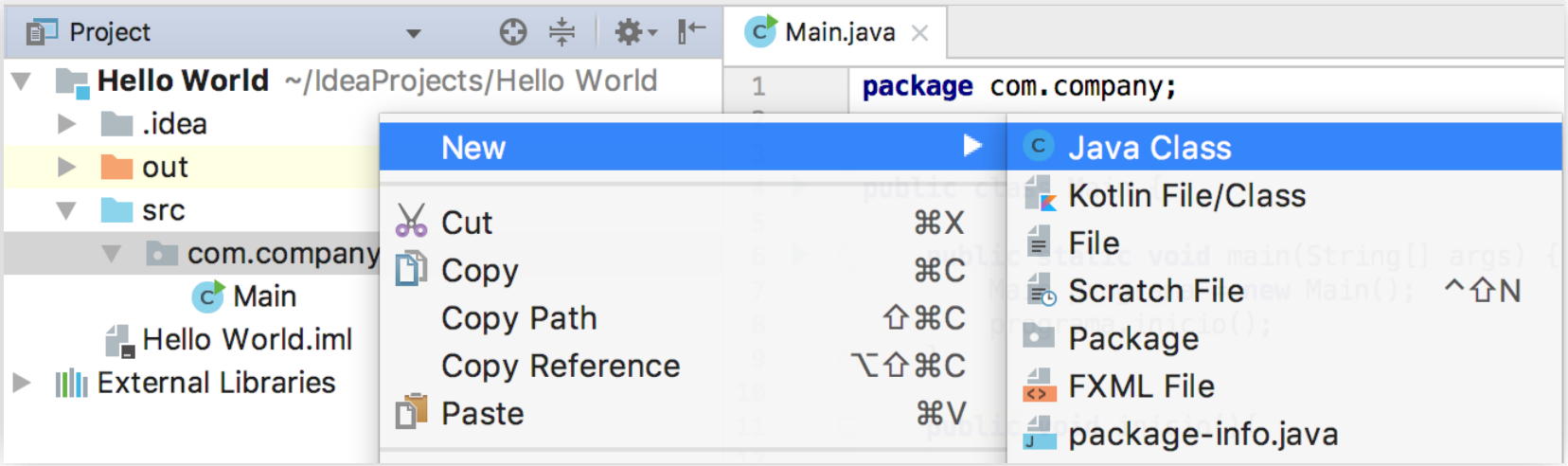
Project

- Hello World ~/IdeaProjects/Hello World
  - .idea
  - out
  - src
    - com.company
      - Main
  - Hello World.iml
- External Libraries

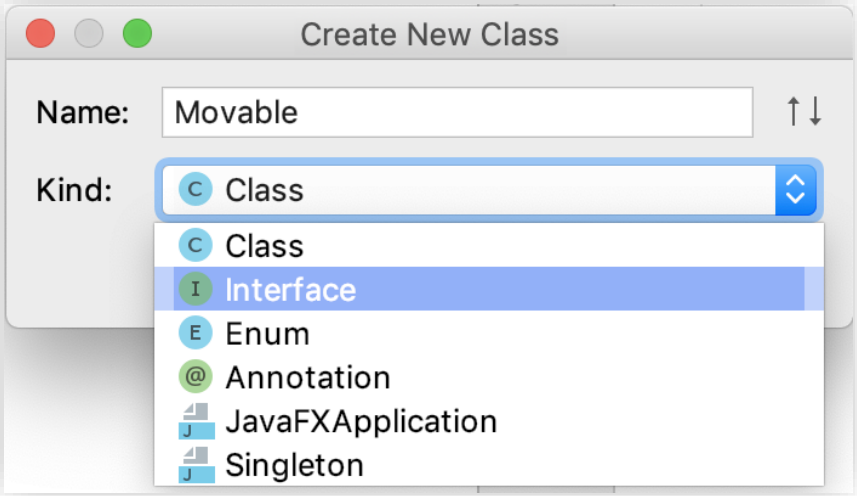
Main.java

```
1   package com.company;
2
3
4   public class Main {
5
6       public static void main(String[] args) {
7           Main programa = new Main();
8           programa.inicio();
9       }
10
11      public void inicio(){
12
13          System.out.println("Hello World");
14
15      }
16
17  }
```

3. Creating an interface in IntelliJ



3. Creating an interface in IntelliJ



3. Creating an interface in IntelliJ

New File:
Movable.java

HelloWorld > src > com > company > Movable

1: Project

Project

HelloWorld ~/IdeaProjects/HelloWorld
- .idea
- out
- src
  - com.company
    - Main
    - Movable
  - HelloWorld.iml
- External Libraries

Main.java | Movable.java

```
1  package com.company;
2
3  public interface Movable {
4  }
5
```

Java predefined interfaces

3. Java predefined interfaces

Module java.base
Package java.lang

Interface Comparable<T>

Type Parameters:

T - the type of objects that this object may be compared to

docs.oracle.com

3. Java predefined interfaces

Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method	Description
`int`	`compareTo(T o)`	Compares this object with the specified object for order.

docs.oracle.com

3. Java predefined interfaces

```
public class Book{

  private int numberOfPages;
  private String title;
  private String author;

  public Book(int numberOfPages, String title, String author) {
   this.numberOfPages = numberOfPages;
   this.title = title;
   this.author = author;
  }

}
```

Book.java

3. Java predefined interfaces

```
import java.util.ArrayList;
import java.util.Collections;

public class Main{

   public static void main(String[] args) {

      ArrayList<Book> books = ...
      Collections.sort(books);

   }
}
```

Order criteria:

1. Sort by the title alphabetically
2. If both books have the same title, then sort by the author alphabetically
3. If both books have the same title and author, then sort by number of pages

Main.java

3. Java predefined interfaces

```
import java.util.ArrayList;
import java.util.Collections;

public class Main{

   public static void main(String[] args) {

      ArrayList<Book> books = ...
      Collections.sort(books);

   }
}
```

Book class needs to implement Comparable interface

Main.java

3. Java predefined interfaces

```
public class Book{

  private int numberOfPages;
  private String title;
  private String author;

  public Book(int numberOfPages, String title, String author) {
   this.numberOfPages = numberOfPages;
   this.title = title;
   this.author = author;
  }

}
```

Book.java

3. Java predefined interfaces

```
public class Book implements Comparable{

  private int numberOfPages;
  private String title;
  private String author;

  public Book(int numberOfPages, String title, String author) {
   this.numberOfPages = numberOfPages;
   this.title = title;
   this.author = author;
  }

}
```

Book.java

3. Java predefined interfaces

```
public class Book implements Comparable{

  private int numberOfPages;
  private String title;
  private String author;

  public Book(int numberOfPages, String title, String author) {...}

  @Override
  public int compareTo(Book o) {
   return 0;
  }
}
```

Book.java

3. Java predefined interfaces

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.

The implementor must ensure `sgn(x.compareTo(y)) == -sgn(y.compareTo(x))` for all `x` and `y`. (This implies that `x.compareTo(y)` must throw an exception iff `y.compareTo(x)` throws an exception.)

The implementor must also ensure that the relation is transitive: `(x.compareTo(y) > 0 && y.compareTo(z) > 0)` implies `x.compareTo(z) > 0`.

Finally, the implementor must ensure that `x.compareTo(y)==0` implies that `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`, for all `z`.

It is strongly recommended, but *not* strictly required that `(x.compareTo(y)==0) == (x.equals(y))`. Generally speaking, any class that implements the `Comparable` interface and violates this condition should clearly indicate this fact. The recommended language is "Note: this class has a natural ordering that is inconsistent with equals."

In the foregoing description, the notation `sgn(`*expression*`)` designates the mathematical *signum* function, which is defined to return one of `-1`, `0`, or `1` according to whether the value of *expression* is negative, zero, or positive, respectively.

Parameters:

`o` - the object to be compared.

Returns:

a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

docs.oracle.com

3. Java predefined interfaces

```
public class Book implements Comparable{

  private int numberOfPages;
  private String title;
  private String author;

  public Book(int numberOfPages, String title, String author) {...}

  @Override
  public int compareTo(Book o) {

    if (!this.title.equalsIgnoreCase(o.title)){
      return this.title.compareTo(o.title);
      //If titles are identical, lets compare authors
    }else if (!this.author.equalsIgnoreCase(o.author)){
       return this.author.compareTo(o.author);
     //If titles and authors are identical, lets compare number of pages
    }else{
      return this.numberOfPages-o.numberOfPages;
    }

  }
}
```

Book.java

“El único lugar donde el éxito está antes que el trabajo, es en el diccionario.“

Vince Lombardi, considerado uno de los mejores entrenadores de futbol americano