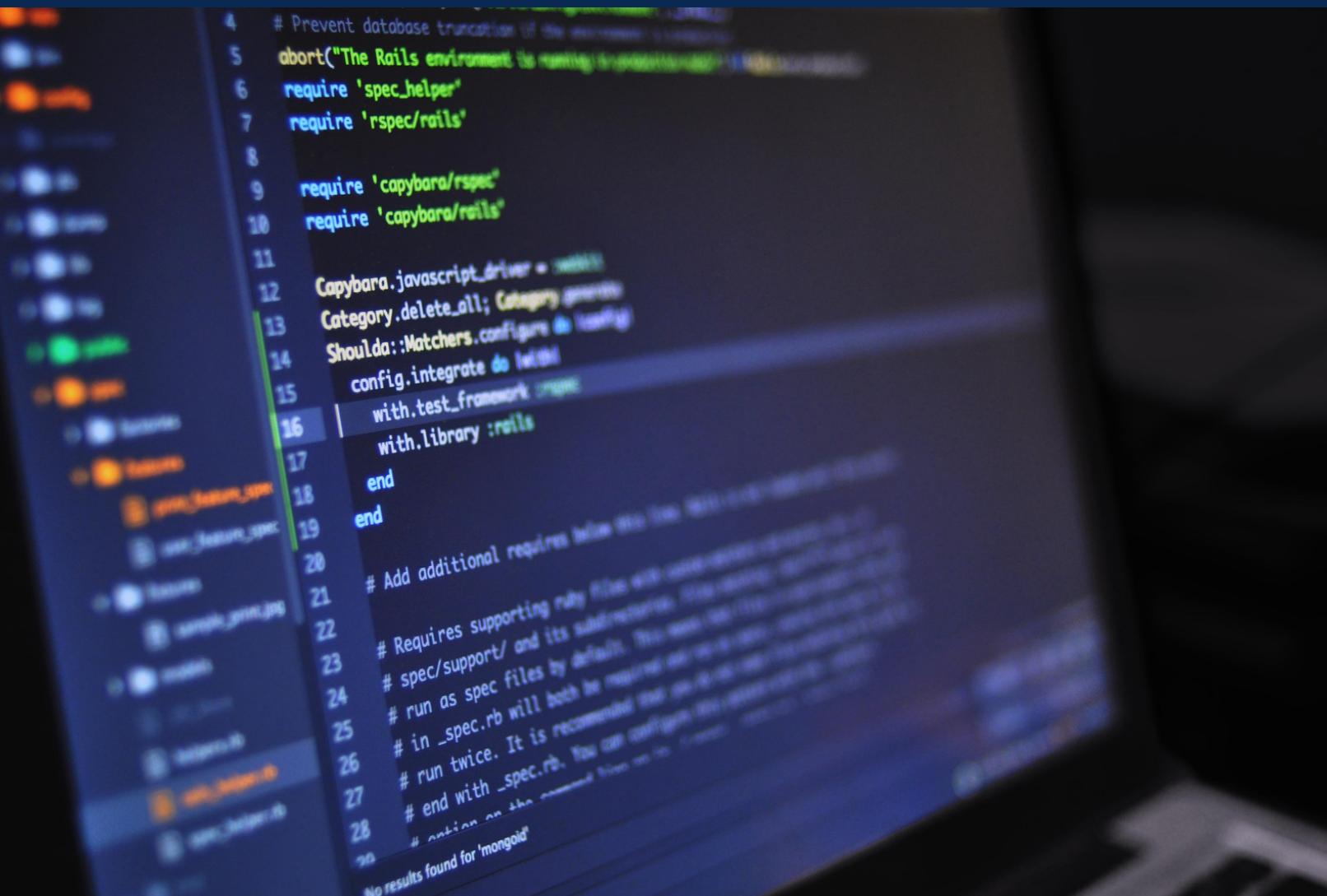


TAREA FINAL

PROGRAMACIÓN DE SERVICIOS Y PROCESOS

VICTOR, ÁLVARO, IRENE, MARTA

ÍNDICE DE CONTENIDOS



```
4 # Prevent database truncation if the environment is test or eval
5 abort("The Rails environment is running in production mode")
6 require 'spec_helper'
7 require 'rspec/rails'
8
9 require 'capybara/rspec'
10 require 'capybara/rails'
11
12 Capybara.javascript_driver = :webkit
13 Category.delete_all; Category.create!(name: "Electronics")
14 Shoulda::Matchers.configure do |config|
15   config.integrate do |with|
16     with.test_framework :rspec
17     with.library :rails
18   end
19 end
20
21 # Add additional requires below this line if you need them
22
23 # Requires supporting ruby files with custom matchers and helpers
24 # in spec/support/ and its subdirectories. This directory also
25 # contains supporting files for this generator; custom generators and
26 # helper macros can be added in .generator and .macro files respectively
27 # run as spec files by default. You can change this behavior by
28 # specifying :type, for example:
# specify :type, :unit
# run twice. It is recommended you do not name your
# tasks with the suffix _spec.rb. You can configure the
# suffix via the --suffix command line option.
# or in config/generator.rb
# or in config/macro.rb
# no results found for 'mongoid'
```

0. INTRODUCCIÓN

1. DIAGRAMA UML

2. CONSTRUCTORES

2.1. VACÍO

2.2. PARAMETRIZADO

2.3. DE HERENCIA

3. CASTING

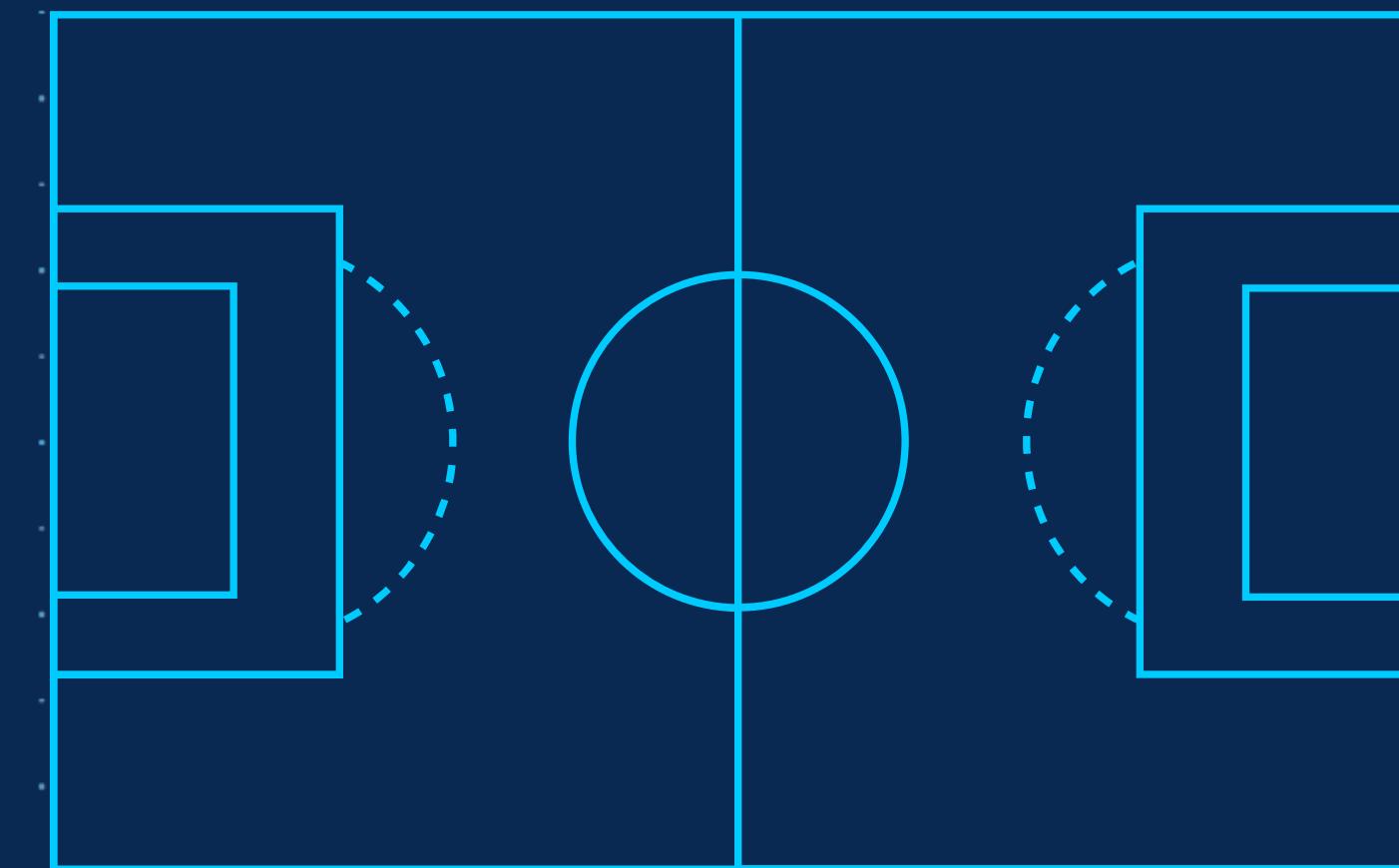
4. OVERLOADING

5. OVERRIDING

6. RELACIONES ENTRE CLASES

TAREA FINAL

0. INTRODUCCIÓN





APLICACIÓN DE FÚTBOL

EN ESTE PROYECTO, HEMOS DESARROLLADO UNA APLICACIÓN DE FÚTBOL EN JAVA QUE SIMULA PARTIDOS ENTRE EQUIPOS.

LA APLICACIÓN SE CENTRA EN LAS CLASES ESTADIO, EQUIPO, JUGADOR, Y PARTIDO, QUE JUNTAS PROPORCIONAN UN MARCO PARA ORGANIZAR Y JUGAR PARTIDOS DE FÚTBOL.



1. DIAGRAMA UML

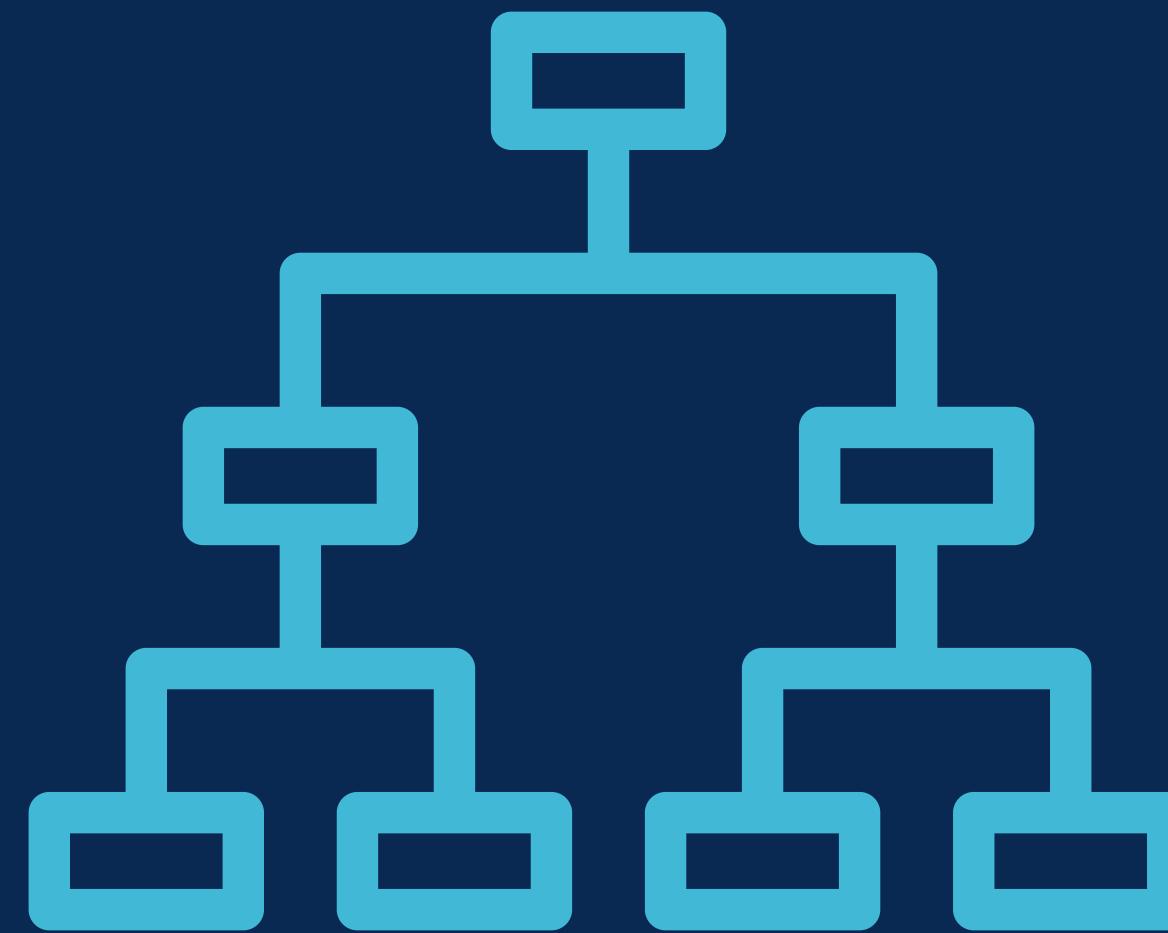
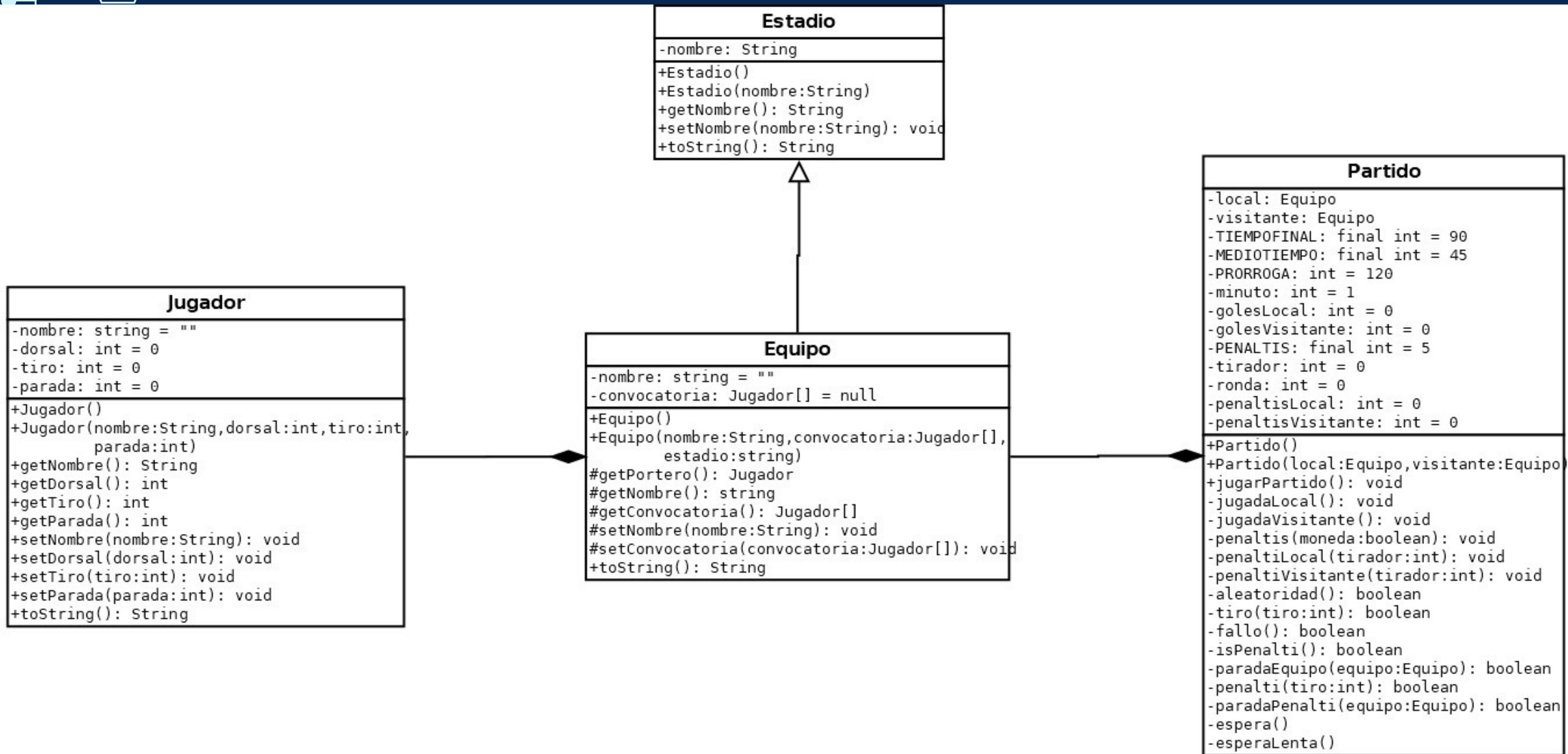


DIAGRAMA UML



2. CONSTRUCTORES



¿CUÁL ES LA FINALIDAD DE LOS CONSTRUCTORES?

LOS CONSTRUCTORES SON FUNDAMENTALES PARA LA CREACIÓN Y CONFIGURACIÓN DE OBJETOS EN JAVA Y OTROS LENGUAJES ORIENTADOS A OBJETOS.

LA SOBRECARGA DE CONSTRUCTORES AUMENTA LA FLEXIBILIDAD Y LA ADAPTABILIDAD DEL CÓDIGO.

```
public class Estadio {  
  
    private String nombre;  
  
    public Estadio() {  
        this.nombre = "";  
    }  
  
    public Estadio(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

```
public class Equipo extends Estadio {  
  
    // Atributos del equipo  
  
    private String nombre;  
    private Jugador[] convocatoria;  
  
    // Constructores (Overloading)  
  
    public Equipo() {  
        super();  
        this.nombre = "";  
        this.convocatoria = null;  
    }  
  
    public Equipo(String nombre, String estadio, Jugador[] convocatoria) {  
        super(estadio);  
        this.nombre = nombre;  
        this.convocatoria = convocatoria;  
    }  
}
```

```
// Constructores (Overloading)
public Jugador() {
    this.nombre = "";
    this.dorsal = 0;
    this.tiro = 0;
    this.parada = 0;
}

public Jugador(String nombre, int dorsal, int tiro, int parada) {
    this.nombre = nombre;
    this.dorsal = dorsal;
    this.tiro = tiro;
    this.parada = parada;
}
```

```
// Constructores (Overloading)
public Partido() {
    this.local = null;
    this.visitante = null;
}

public Partido(Equipo local, Equipo visitante) {
    this.local = local;
    this.visitante = visitante;
}
```

3. CASTING



¿QUÉ ES EL CASTING EN JAVA?

EL CASTING EN JAVA SE REFIERE A LA CONVERSIÓN DE UN TIPO DE DATO A OTRO. PUEDE SER EXPLÍCITO (FORZADO POR EL PROGRAMADOR) O IMPLÍCITO (REALIZADO AUTOMÁTICAMENTE POR EL COMPILADOR).

EL CASTING ES ESENCIAL PARA MANIPULAR TIPOS DE DATOS EN DIFERENTES CONTEXTOS, ESPECIALMENTE CUANDO SE TRABAJA CON HERENCIA Y POLIMORFISMO.

PERMITE ADAPTAR DATOS DE UN TIPO A OTRO DE MANERA SEGURA Y EFICIENTE.

CASTING EXPLÍCITO EN EL CÓDIGO

```
// Métodos de aleatoriedad

private boolean aleatoriedad() {
    return ((int)(Math.random() * 100)) % 2 == 0;
}

private boolean tiro(int tiro) {
    return (int)(Math.random() * 100) < tiro && fallo();
}
```

```
private boolean penalti(int tiro) {  
    return (int)(Math.random() * 100) < (tiro * 2);  
}  
  
private boolean paradaPenalti(Equipo equipo) {  
    return (int)(Math.random() * 100) <= equipo.getPortero().getParada() && aleatoriedad();  
}
```

4. OVERLOADING



¿QUÉ ES LA SOBRECARGA DE MÉTODOS?

LA SOBRECARGA DE MÉTODOS EN JAVA PERMITE QUE UNA CLASE TENGA MÚLTIPLES MÉTODOS CON EL MISMO NOMBRE, PERO CON DIFERENTES PARÁMETROS.

LA SOBRECARGA DE CONSTRUCTORES PROPORCIONA FLEXIBILIDAD AL PERMITIR LA CREACIÓN DE OBJETOS “EQUIPO” EN DIVERSAS SITUACIONES.

PERMITE QUE LOS DESARROLLADORES UTILICEN EL CONSTRUCTOR QUE MEJOR SE ADAPTE A SUS NECESIDADES SIN TENER QUE RECORDAR MÚLTIPLES NOMBRES DE MÉTODO.

```
public class Equipo extends Estadio {  
  
    // Atributos del equipo  
  
    private String nombre;  
    private Jugador[] convocatoria;  
  
    // Constructores (Overloading)  
  
    public Equipo() {  
        super();  
        this.nombre = "";  
        this.convocatoria = null;  
    }  
  
    public Equipo(String nombre, String estadio, Jugador[] convocatoria) {  
        super(estadio);  
        this.nombre = nombre;  
        this.convocatoria = convocatoria;  
    }  
}
```

5. OVERRIDING



¿QUÉ ES LA SOBRESCRITURA DE MÉTODOS EN JAVA?

LA SOBRESCRITURA DE MÉTODOS EN JAVA PERMITE A UNA SUBCLASE PROPORCIONAR UNA IMPLEMENTACIÓN ESPECÍFICA DE UN MÉTODO QUE YA ESTÁ DEFINIDO EN SU CLASE BASE. EN ESTE CASO, NOS ENFOCAREMOS EN LA SOBRESCRITURA DEL MÉTODO `toString`

EL MÉTODO `TOSTRING` SE UTILIZA PARA PROPORCIONAR UNA REPRESENTACIÓN LEGIBLE EN CADENA DE UN OBJETO.

LA SOBRESCRITURA PERMITE PERSONALIZAR LA SALIDA DE CADENA PARA ADAPTARSE A LAS NECESIDADES ESPECÍFICAS DE LAS CLASES EQUIPO.

```
public class Equipo extends Estadio {  
  
    // Atributos del equipo  
  
    private String nombre;  
    private Jugador[] convocatoria;  
  
    // toString (Overriding)  
  
    @Override  
    public String toString() {  
  
        String listaJugadores = "";  
  
        for (Jugador jugador : convocatoria) {  
            listaJugadores += "\n" + jugador.toString();  
        }  
  
        return "\n" + getNombre() + super.toString() + listaJugadores;  
    }  
}
```

6. RELACIONES ENTRE CLASES (COMPOSICIÓN)



¿QUÉ ES LA COMPOSICIÓN EN JAVA?

LA COMPOSICIÓN ES UNA RELACIÓN ENTRE CLASES EN LA QUE UN OBJETO CONTIENE OTRO OBJETO, LO QUE IMPLICA QUE LA EXISTENCIA DEL OBJETO CONTENIDO DEPENDE DEL OBJETO QUE LO CONTIENE.

LA COMPOSICIÓN PERMITE QUE LA CLASE “PARTIDO” ESTÉ COMPUESTA POR UN ATRIBUTO QUE A SU VEZ ES OTRA CLASE “EQUIPO”

CREANDO ASÍ UNA RELACIÓN ESTRUCTURADA.

LA RELACIÓN SE ESTABLECE POR LOS DATOS QUE SE TRANSMITEN.

```
public class Partido {  
    // Atributos del partido  
    private Equipo local, visitante;  
  
    public Partido() {  
        this.local = null;  
        this.visitante = null;  
    }  
  
    public Partido(Equipo local, Equipo visitante) {  
        this.local = local;  
        this.visitante = visitante;  
    }  
}
```

LA CLASE "EQUIPO" TIENE UNA VARIABLE CONVOCATORIA QUE ES UN ARRAY DE OBJETOS DE LA CLASE "JUGADOR"

```
public class Equipo extends Estadio {  
    // Atributos del equipo  
    private String nombre;  
    private Jugador[] convocatoria;
```

```
public Equipo() {  
    super();  
    this.nombre = "";  
    this.convocatoria = null;  
}  
  
public Equipo(String nombre, String estadio, Jugador[] convocatoria) {  
    super(estadio);  
    this.nombre = nombre;  
    this.convocatoria = convocatoria;  
}
```

```
    render() {
      return (
        <React.Fragment>
          <div className="py-5">
            <div className="container">
              <Title name="our" title="product">
              <div className="row">
                <ProductConsumer>
                  {(value) => {
                    |   |   |   console.log(value)
                    |   |   |
                  }}
                </ProductConsumer>
              </div>
            </div>
          </div>
        <React.Fragment>
```

| MUCHAS GRACIAS !