

Perfecto, pues vamos a ello 😊

Te dejo **solo los archivos que realmente hace falta tocar/crear** para que:

- El JAXB de corredores/equipos cuadre con `Corredores.xml` y `Equipos.xml`.
- El ejercicio de **Registro de personas** genere un XML como el del enunciado.

Voy por bloques.

---

## 1. CORREDORES / EQUIPOS (A4UD1)

### 1.1. `Velocista.java` ✅ (corregir constructores)

```
package clases.ClasesCorredor;

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlElement;
import jakarta.xml.bind.annotation.XmlType;

import java.time.LocalDate;
import java.util.List;

/**
 * Clase que representa a un Velocista.
 * Está pensada para ser usada dentro del elemento raíz <corredores>.
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(propOrder = {"nombre", "fechaNacimiento", "velocidadMedia", "historial"})
public class Velocista extends Corredor {

    @XmlElement(name = "velocidad_media")
    private float velocidadMedia;

    /**
     * Constructor completo usado en la lógica del programa.
     * IMPORTANTE: ahora sí llama al constructor de la superclase para
     * rellenar código, dorsal, nombre, equipo y fecha.
     */
    public Velocista(String codigo,
                     String nombre,
                     String equipo,
                     int dorsal,
                     LocalDate fecha,
                     List<Puntuacion> historial,
                     float velocidad) {
        // Inicializamos los campos heredados de Corredor
        super(codigo, nombre, equipo, dorsal, fecha, historial);
        this.velocidadMedia = velocidad;
    }

    /**
     * Constructor sin historial, por si lo quieres usar para crear
     * corredores que todavía no tienen puntuaciones.
     */
    public Velocista(String codigo,
                     String nombre,
                     String equipo,
                     int dorsal,
                     LocalDate fecha,
```

```

        float velocidad) {
            super(codigo, nombre, equipo, dorsal, fecha);
            this.velocidadMedia = velocidad;
        }

/**
 * Constructor vacío obligatorio para JAXB.
 */
public Velocista() {}

public float getVelocidadMedia() {
    return velocidadMedia;
}

public void setVelocidadMedia(float velocidadMedia) {
    this.velocidadMedia = velocidadMedia;
}

@Override
public String toString() {
    StringBuilder sb = new StringBuilder(super.toString());

    sb.append(" | VELOCIDAD MEDIA: ");
    sb.append(String.format("%.2f km/h", this.velocidadMedia));

    return sb.toString();
}
}

```

---

## 1.2. Fondista.java (constructores igual que Velocista)

```

package clases.ClasesCorredor;

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlElement;
import jakarta.xml.bind.annotation.XmlType;

import java.time.LocalDate;
import java.util.List;

/**
 * Clase que representa a un Fondista.
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(propOrder = {"nombre", "fechaNacimiento", "distanciaMax", "historial"})
public class Fondista extends Corredor {

    @XmlElement(name = "distancia_max")
    private float distanciaMax;

/**
 * Constructor completo, ahora llamando a super(...)
 * para inicializar los datos comunes del corredor.
 */
    public Fondista(String codCorredor,
                    String nombre,
                    String equipo,
                    int dorsal,
                    LocalDate fecha,
                    List<Puntuacion> historial,
                    float distancia) {
        super(codCorredor, nombre, equipo, dorsal, fecha, historial);
        this.distanciaMax = distancia;
    }
}

```

```

    }

    /**
     * Constructor sin historial, por si lo necesitas.
     */
    public Fondista(String codigo,
                    String nombre,
                    String equipo,
                    int dorsal,
                    LocalDate fecha,
                    float distancia) {
        super(codigo, nombre, equipo, dorsal, fecha);
        this.distanciaMax = distancia;
    }

    /**
     * Constructor vacío para JAXB.
     */
    public Fondista() {}

    public float getDistanciaMax() {
        return distanciaMax;
    }

    public void setDistanciaMax(float distanciaMax) {
        this.distanciaMax = distanciaMax;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder(super.toString());
        sb.append(" | DISTANCIA MAX: ");
        sb.append(String.format("%.3f km", this.distanciaMax));
        return sb.toString();
    }
}

```

---

### 1.3. Equipo.java (setters vacíos → implementados + getter patrocinadores)

```

package clases.ClasesEquipo;

import jakarta.xml.bind.annotation.*;

/**
 * Representa un <equipo> del XML equipos.xml.
 */
@XmlAccessorType(XmlAccessType.FIELD)
public class Equipo {

    @XmlAttribute(name = "id", required = true)
    private String idEquipo;

    @XmlElement(name = "nombre", required = true)
    private String nombre;

    @XmlElement(name = "patrocinadores", required = true)
    private Patrocinadores patrocinadores;

    /**
     * Constructor vacío para JAXB.
     */
    public Equipo() {}

    public Equipo(String nombre) {

```

```

        this.nombre = nombre;
    }

    public Equipo(String idEquipo, String nombreEquipo) {
        this.idEquipo = idEquipo;
        this.nombre = nombreEquipo;
    }

    public String getIdEquipo() {
        return idEquipo;
    }

    public void setIdEquipo(String idEquipo) {
        this.idEquipo = idEquipo;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public Patrocinadores getPatrocinadores() {
        return patrocinadores;
    }

    public void setPatrocinadores(Patrocinadores patrocinadores) {
        this.patrocinadores = patrocinadores;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Equipo{");
        sb.append("idEquipo=").append(idEquipo);
        sb.append(", nombreEquipo=").append(nombre).append('\n');
        sb.append(", patrocinadores=").append(patrocinadores);
        return sb.append("}").toString();
    }
}

```

---

#### 1.4. Patrocinadores. java (getter/setter del set + ayuda con numPatrocinadores)

```

package clases.ClasesEquipo;

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlAttribute;
import jakarta.xml.bind.annotation.XmlElement;

import java.util.HashSet;
import java.util.Set;

/**
 * Contenedor de varios <patrocinador>.
 * Mapea al elemento <patrocinadores numPatrocinadores="..."> del XML.
 */
@XmlAccessorType(XmlAccessType.FIELD)
public class Patrocinadores {

    @XmlAttribute(name = "numPatrocinadores", required = true)
    private int numPatrocinadores;
}

```

```

@XmlElement(name = "patrocinador", required = true)
private Set<Patrocinador> patrocinadores = new HashSet<>();

public Patrocinadores() {}

public int getNumPatrocinadores() {
    return numPatrocinadores;
}

public void setNumPatrocinadores(int numPatrocinadores) {
    this.numPatrocinadores = numPatrocinadores;
}

public Set<Patrocinador> getPatrocinadores() {
    return patrocinadores;
}

public void setPatrocinadores(Set<Patrocinador> patrocinadores) {
    this.patrocinadores = patrocinadores != null ? patrocinadores : new HashSet<>();
    // opcional: sincronizar el atributo con el tamaño del set
    this.numPatrocinadores = this.patrocinadores.size();
}

/**
 * Método de ayuda para añadir un patrocinador y actualizar el contador.
 */
public void addPatrocinador(Patrocinador p) {
    if (p != null) {
        this.patrocinadores.add(p);
        this.numPatrocinadores = this.patrocinadores.size();
    }
}

@Override
public String toString() {
    return "Patrocinadores{" +
        "numPatrocinadores=" + numPatrocinadores +
        ", patrocinadores=" + patrocinadores +
        '}';
}
}

```

## 1.5. Patrocinador.java (quitar static y dejarlo normal de instancia)

```
package clases.ClasesEquipo;
```

```

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlAttribute;
import jakarta.xml.bind.annotation.XmlValue;
import jakarta.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
import persistencia.LocalDateAdapter;

import java.time.LocalDate;
import java.util.Objects;

/**
 * Representa un <patrocinador> dentro de <patrocinadores>.
 * Ejemplo en XML:
 * <patrocinador donacion="1500.0" fecha_inicio="2023-01-15">Nike</patrocinador>
 */
@XmlAccessorType(XmlAccessType.FIELD)
public class Patrocinador {

```

```

// Texto entre etiquetas: nombre del patrocinador
@XmlValue
private String nombre;

// Atributo donacion="..."
@XmlAttribute(name = "donacion", required = true)
private float donacion;

// Atributo fecha_inicio="yyyy-MM-dd"
@XmlAttribute(name = "fecha_inicio", required = true)
@XmlJavaTypeAdapter(LocalDateAdapter.class)
private LocalDate fechaInicio;

/**
 * Constructor vacío para JAXB.
 */
public Patrocinador() {}

public Patrocinador(String nombrePatrocinador, float donacion, LocalDate fechaInicio) {
    this.nombre = nombrePatrocinador;
    this.donacion = donacion;
    this.fechaInicio = fechaInicio;
}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public float getDonacion() {
    return donacion;
}

public void setDonacion(float donacion) {
    this.donacion = donacion;
}

public LocalDate getFechaInicio() {
    return fechaInicio;
}

public void setFechaInicio(LocalDate fechaInicio) {
    this.fechaInicio = fechaInicio;
}

/**
 * Igualamos patrocinadores por nombre para poder usar Set y evitar duplicados.
 */
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Patrocinador that)) return false;
    return Objects.equals(nombre, that.nombre);
}

@Override
public int hashCode() {
    return Objects.hashCode(nombre);
}

@Override

```

```

    public String toString() {
        return String.format("%s | Donación: %.2f | Inicio: %s",
            nombre,
            donacion,
            fechaInicio);
    }
}

```

---

## 1.6. EquiposJAXB.java (añadir método para escribir XML)

```

package persistencia;

import clases.ClasesEquipo.Equipos;
import jakarta.xml.bind.JAXBException;

/**
 * Clase de acceso a la persistencia XML de equipos usando JAXB.
 */
public class EquiposJAXB {

    /**
     * Lee el XML de equipos y lo convierte en un objeto Equipos.
     */
    public static Equipos leerEquipos(String rutaXML) throws JAXBException {
        try {
            return XMLJAXBUtils.unmarshal(Equipos.class, rutaXML);
        } catch (JAXBException e) {
            throw new JAXBException("Error al leer el XML de equipos: " + rutaXML, e);
        }
    }

    /**
     * NUEVO:
     * Escribe un objeto Equipos en la ruta indicada.
     * Te servirá, por ejemplo, para guardar EquiposUpdateJAXB.xml.
     */
    public static void escribirEquipos(String rutaXML, Equipos equipos) throws JAXBException {
        try {
            XMLJAXBUtils.marshal(equipos, rutaXML);
        } catch (JAXBException e) {
            throw new JAXBException("Error al escribir el XML de equipos: " + rutaXML, e);
        }
    }
}

```

Los de corredores (`Corredor`, `Corredores`, `Puntuacion`, `LocalDateAdapter`) están ya bien alineados con `Corredores.xml`; solo tocamos los constructores de `Velocista` y `Fondista`.

---

## 2. REGISTRO DE PERSONAS (A5UD1)

Aquí sí faltaban bastantes cosas, así que:

- Creamos un **adaptador de fechas específico** para este ejercicio (`dd-MM-yy`).
- Creamos un **adaptador de categorías** para mapear `List<String>` ↔ texto con espacios.
- Completamos `Registro`, `Persona`, `Trabajador`, `Estudiante`, `Empresa`, `Telefonos`.
- Añadimos **dos clases nuevas**: `Email` y `Telefono`.

## 2.1. NUEVO: LocalDateAdapterRegistro.java (en persistencia)

```
package persistencia;

import jakarta.xml.bind.annotation.adapters.XmlAdapter;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

/**
 * Adaptador específico para el ejercicio de Registro de personas.
 * Convierte entre LocalDate y cadenas con formato dd-M-yy (ej: 12-3-99).
 */
public class LocalDateAdapterRegistro extends XmlAdapter<String, LocalDate> {

    private final DateTimeFormatter formatter = DateTimeFormatter.ofPattern("d-M-yy");

    @Override
    public LocalDate unmarshal(String v) {
        if (v == null || v.isEmpty()) {
            return null;
        }
        return LocalDate.parse(v, formatter);
    }

    @Override
    public String marshal(LocalDate v) {
        if (v == null) {
            return null;
        }
        return v.format(formatter);
    }
}
```

---

## 2.2. NUEVO: CategoriasAdapter.java (en persistencia)

```
package persistencia;

import jakarta.xml.bind.annotation.adapters.XmlAdapter;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

/**
 * Adaptador para mapear List<String> <-> "Estudios Trabajo Deporte"
 * en el elemento <Categorias> del XML.
 */
public class CategoriasAdapter extends XmlAdapter<String, List<String>> {

    @Override
    public List<String> unmarshal(String v) {
        if (v == null || v.isBlank()) {
            return Collections.emptyList();
        }
        // Separar por espacios
        return Arrays.asList(v.trim().split("\\s+"));
    }

    @Override
    public String marshal(List<String> v) {
        if (v == null || v.isEmpty()) {
            return "";
        }
    }
}
```



```

        // Unir categorías separadas por espacios
        return String.join(" ", v);
    }
}

```

---

### 2.3. Registro.java (clase raíz del XML)

```

package clases.ClasesRegistro;

import jakarta.xml.bind.annotation.*;
import jakarta.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
import persistencia.CategoriasAdapter;
import persistencia.LocalDateAdapterRegistro;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

/**
 * Clase raíz que representa el elemento <Registro> del XML.
 *
 * Ejemplo:
 * <Registro version="1.0" fechaCreacion="17-11-25">
 *     <Categorias>Estudios Trabajo Deporte</Categorias>
 *     <Personas>
 *         ...
 *     </Personas>
 * </Registro>
 */
@XmlRootElement(name = "Registro")
@XmlAccessorType(XmlAccessType.FIELD)
public class Registro {

    @XmlAttribute(name = "version")
    private String version;

    @XmlAttribute(name = "fechaCreacion")
    @XmlJavaTypeAdapter(LocalDateAdapterRegistro.class)
    private LocalDate fechaCreacion;

    @XmlElement(name = "Categorias")
    @XmlJavaTypeAdapter(CategoriasAdapter.class)
    private List<String> categorias = new ArrayList<>();

    @XmlElementWrapper(name = "Personas")
    @XmlElements({
        @XmlElement(name = "Trabajador", type = Trabajador.class),
        @XmlElement(name = "Estudiante", type = Estudiante.class)
    })
    private List<Persona> personas = new ArrayList<>();

    public Registro() {}

    public String getVersion() {
        return version;
    }

    public void setVersion(String version) {
        this.version = version;
    }

    public LocalDate getFechaCreacion() {
        return fechaCreacion;
    }
}

```

```

    public void setFechaCreacion(LocalDate fechaCreacion) {
        this.fechaCreacion = fechaCreacion;
    }

    public List<String> getCategorias() {
        return categorias;
    }

    public void setCategorias(List<String> categorias) {
        this.categorias = categorias != null ? categorias : new ArrayList<>();
    }

    public List<Persona> getPersonas() {
        return personas;
    }

    public void setPersonas(List<Persona> personas) {
        this.personas = personas != null ? personas : new ArrayList<>();
    }

    public void addPersona(Persona p) {
        if (p != null) {
            this.personas.add(p);
        }
    }
}

```

---

## 2.4. Persona . java (usar adaptador de fecha correcto y contacto polimórfico)

```

package clases.ClasesRegistro;

import jakarta.xml.bind.annotation.*;
import jakarta.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
import persistencia.LocalDateAdapterRegistro;

import java.time.LocalDate;

/**
 * Clase abstracta base para Trabajador y Estudiante.
 * No se genera etiqueta <Persona>, solo <Trabajador> y <Estudiante>.
 */
@XmlAccessorType(XmlAccessType.NONE)
@XmlSeeAlso({Trabajador.class, Estudiante.class})
@XmlTransient
public abstract class Persona {

    @XmlElement(name = "Nombre")
    private String nombre;

    @XmlElement(name = "FechaNacimiento")
    @XmlJavaTypeAdapter(LocalDateAdapterRegistro.class)
    private LocalDate fechaNacimiento;

    /**
     * Contacto polimórfico:
     * - <Telefonos> ... </Telefonos> -> clase Telefonos
     * - <Email>texto</Email> -> clase Email
     * Solo debe aparecer uno de los dos en cada Persona.
     */
    @XmlElements({
        @XmlElement(name = "Telefonos", type = Telefonos.class),
        @XmlElement(name = "Email", type = Email.class)
    })
}

```

```

    private Contacto contacto;

    public Persona() {}

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public LocalDate getFechaNacimiento() {
        return fechaNacimiento;
    }

    public void setFechaNacimiento(LocalDate fechaNacimiento) {
        this.fechaNacimiento = fechaNacimiento;
    }

    public Contacto getContacto() {
        return contacto;
    }

    public void setContacto(Contacto contacto) {
        this.contacto = contacto;
    }
}

```

---

## 2.5. Trabajador.java

```

package clases.ClasesRegistro;

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlElement;

/**
 * Representa un <Trabajador> del XML.
 *
 * <Trabajador>
 *   <Nombre>...</Nombre>
 *   <FechaNacimiento>...</FechaNacimiento>
 *   <Telefonos>... o <Email>...</Email>
 *   <Empresa puesto="...">NombreEmpresa</Empresa>
 *   <Salario>...</Salario>
 * </Trabajador>
 */
@XmlAccessorType(XmlAccessType.FIELD)
public class Trabajador extends Persona {

    @XmlElement(name = "Empresa")
    private Empresa empresa;

    @XmlElement(name = "Salario")
    private double salario;

    public Trabajador() {}

    public Empresa getEmpresa() {
        return empresa;
    }

    public void setEmpresa(Empresa empresa) {

```

```

        this.empresa = empresa;
    }

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }
}

```

---

## 2.6. Estudiante.java ✓

```

package clases.ClasesRegistro;

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlElement;

/**
 * Representa un <Estudiante> del XML.
 *
 * <Estudiante>
 *   <Nombre>...</Nombre>
 *   <FechaNacimiento>...</FechaNacimiento>
 *   <Email>...</Email> o <Telefonos>...</Telefonos>
 *   <Universidad>...</Universidad>
 *   <Carrera>...</Carrera>
 * </Estudiante>
 */
@XmlAccessorType(XmlAccessType.FIELD)
public class Estudiante extends Persona {

    @XmlElement(name = "Universidad")
    private String universidad;

    @XmlElement(name = "Carrera")
    private String carrera;

    public Estudiante() {}

    public String getUniversidad() {
        return universidad;
    }

    public void setUniversidad(String universidad) {
        this.universidad = universidad;
    }

    public String getCarrera() {
        return carrera;
    }

    public void setCarrera(String carrera) {
        this.carrera = carrera;
    }
}

```

---

## 2.7. Empresa.java ✓ (añadir atributo puesto)

```

package clases.ClasesRegistro;

```

```

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlAttribute;
import jakarta.xml.bind.annotation.XmlValue;

/**
 * Representa el elemento:
 *
 * <Empresa puesto="Director">LaColmena</Empresa>
 */
@XmlAccessorType(XmlAccessType.FIELD)
public class Empresa {

    @XmlAttribute(name = "puesto")
    private String puesto;

    @XmlValue
    private String nombre;

    public Empresa() {}

    public Empresa(String nombre, String puesto) {
        this.nombre = nombre;
        this.puesto = puesto;
    }

    public String getPuesto() {
        return puesto;
    }

    public void setPuesto(String puesto) {
        this.puesto = puesto;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

```

---

## 2.8. Contacto.java (lo dejo casi igual, solo recordatorio)

```

package clases.ClasesRegistro;

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlSeeAlso;

/**
 * Clase base abstracta para los tipos de contacto:
 * - Telefonos
 * - Email
 *
 * No tiene campos propios, solo sirve para el polimorfismo en Persona.
 */
@XmlAccessorType(XmlAccessType.NONE)
@XmlSeeAlso({Telefonos.class, Email.class})
public abstract class Contacto {

    // No tiene nada, las hijas definen sus propios campos.
}

```

---

## 2.9. Telefonos.java (antes estaba vacío)

```
package clases.ClasesRegistro;

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlElement;

import java.util.ArrayList;
import java.util.List;

/**
 * Representa el bloque:
 *
 * <Telefonos>
 *   <Telefono tipo="movil">111111111</Telefono>
 *   <Telefono tipo="fijo">222222222</Telefono>
 * </Telefonos>
 */
@XmlAccessorType(XmlAccessType.FIELD)
public class Telefonos extends Contacto {

    @XmlElement(name = "Telefono")
    private List<Telefono> telefonos = new ArrayList<>();

    public Telefonos() {}

    public List<Telefono> getTelefonos() {
        return telefonos;
    }

    public void setTelefonos(List<Telefono> telefonos) {
        this.telefonos = telefonos != null ? telefonos : new ArrayList<>();
    }

    public void addTelefono(Telefono t) {
        if (t != null) {
            this.telefonos.add(t);
        }
    }
}
```

---

## 2.10. NUEVO: Telefono.java (en clases.ClasesRegistro)

```
package clases.ClasesRegistro;

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlAttribute;
import jakarta.xml.bind.annotation.XmlValue;

/**
 * Elemento individual:
 *
 * <Telefono tipo="movil">111111111</Telefono>
 */
@XmlAccessorType(XmlAccessType.FIELD)
public class Telefono {

    @XmlAttribute(name = "tipo")
    private String tipo;
```

```

@XmlValue
private String numero;

public Telefono() {}

public Telefono(String tipo, String numero) {
    this.tipo = tipo;
    this.numero = numero;
}

public String getTipo() {
    return tipo;
}

public void setTipo(String tipo) {
    this.tipo = tipo;
}

public String getNumero() {
    return numero;
}

public void setNumero(String numero) {
    this.numero = numero;
}
}

```

---

## 2.11. NUEVO: Email.java (en clases.ClasesRegistro)

```

package clases.ClasesRegistro;

import jakarta.xml.bind.annotation.XmlAccessType;
import jakarta.xml.bind.annotation.XmlAccessorType;
import jakarta.xml.bind.annotation.XmlValue;

/**
 * Representa el contacto por email:
 *
 * <Email>correo@example.com</Email>
 */
@XmlAccessorType(XmlAccessType.FIELD)
public class Email extends Contacto {

    @XmlValue
    private String direccion;

    public Email() {}

    public Email(String direccion) {
        this.direccion = direccion;
    }

    public String getDireccion() {
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }
}

```

---



## Qué te quedaría por hacer en tu proyecto

Con estas clases ya puedes:

- Leer `Corredores.xml` y `Equipos.xml` con JAXB correctamente.
- Crear `Corredores` en memoria y escribir `CorredoresNuevos.xml`.
- Leer/escribir `Equipos` para hacer `EquiposUpdateJAXB.xml`.
- Crear un `Registro` con varios `Trabajador` y `Estudiante` y guardarlo en `registrosPersonas.xml` con el formato del enunciado.

Si quieres, en el siguiente mensaje te preparo un **pequeño main** de ejemplo que construya un `Registro` en memoria y lo escriba a `src/test/registrosPersonas.xml` usando tu `XMLJAXBUtils`.

Tú dijiste:

porque no, dale

ChatGPT Plus

Pensó durante 7s

Perfecto, vamos a rematarlo 😊

Te dejo una clase completa con `main` que:

- Crea un `Registro` en memoria.
- Añade:
  - 1 `Trabajador` con **teléfonos y empresa + puesto + salario**.
  - 1 `Estudiante` con **email, universidad y carrera**.
- Rellena `Categorias`.
- Llama a tu `XMLJAXBUtils` para guardar todo en `registrosPersonas.xml`.



### Clase de ejemplo: `GestorRegistro.java` (puede ir en el paquete `logica`)

```
package logica;

import clases.ClasesRegistro.*;
import persistencia.XMLJAXBUtils;
import jakarta.xml.bind.JAXBException;

import java.time.LocalDate;
import java.util.List;

/**
 * Clase de prueba para generar un XML de registros de personas
 * usando las clases JAXB del paquete ClasesRegistro.
 *
 * Al ejecutarla, generará un archivo XML en la ruta indicada
 * con el contenido del objeto Registro creado en memoria.
 */
public class GestorRegistro {

    public static void main(String[] args) {
        try {
            // 1. Crear el objeto Registro (raíz del XML)
            Registro registro = new Registro();

            // Versión del registro (atributo version="1.0")
```



```

registro.setVersion("1.0");

// Fecha de creación del registro (se adaptará a dd-M-yy)
// Ej: LocalDate.of(2025, 11, 17) -> "17-11-25"
registro.setFechaCreacion(LocalDate.of(2025, 11, 17));

// 2. Establecer las categorías (se convertirán a texto separado por espacios)
// <Categorias>Estudios Trabajo Deporte</Categorias>
registro.setCategorias(List.of("Estudios", "Trabajo", "Deporte"));

// 3. Crear un Trabajador con teléfonos
Trabajador trabajador = new Trabajador();
trabajador.setNombre("Carlos Alba");
// Se guardará como "12-3-99" gracias al LocalDateAdapterRegistro
trabajador.setFechaNacimiento(LocalDate.of(1999, 3, 12));

// 3.1. Crear los teléfonos del trabajador
Telefonos telefonos = new Telefonos();
telefonos.addTelefono(new Telefono("movil", "111111111"));
telefonos.addTelefono(new Telefono("fijo", "222222222"));

// Asignar el contacto al trabajador (en este caso, telefonos)
trabajador.setContacto(telefonos);

// 3.2. Crear la empresa con puesto
// <Empresa puesto="Director">LaColmena</Empresa>
Empresa empresa = new Empresa();
empresa.setNombre("LaColmena");
empresa.setPuesto("Director");
trabajador.setEmpresa(empresa);

// 3.3. Salario del trabajador
trabajador.setSalario(8000.0);

// 4. Crear un Estudiante con email
Estudiante estudiante = new Estudiante();
estudiante.setNombre("Ángel Rodríguez");
estudiante.setFechaNacimiento(LocalDate.of(2001, 7, 21));

// 4.1. Email como contacto
Email email = new Email("angel.rodriguez@example.com");
estudiante.setContacto(email);

// 4.2. Universidad y carrera
estudiante.setUniversidad("Santiago");
estudiante.setCarrera("Informática");

// 5. Añadir las personas al registro
registro.addPersona(trabajador);
registro.addPersona(estudiante);

// 6. Guardar el registro en un XML usando XMLJAXBUtils
// Ajusta la ruta si quieres que vaya a otra carpeta
String rutaSalida = "src/test/registrosPersonas.xml";

XMLJAXBUtils.marshal(registro, rutaSalida);

System.out.println("XML de registros de personas generado correctamente en: " + rutaSalida);

} catch (JAXBException e) {
    System.err.println("Error al generar el XML de registros de personas: " + e.getMessage());
    e.printStackTrace();
}
}
}

```

---

## ✅ Checklist rápido

Antes de compilar/ejecutar este `main`, asegúrate de que:

1. Tienes **todas** estas clases tal como te las pasé:
  - `Registro`, `Persona`, `Trabajador`, `Estudiante`, `Contacto`, `Telefonos`, `Telefono`, `Email`, `Empresa`.
  - `LocalDateAdapterRegistro` y `CategoriasAdapter` en persistencia.
2. `XMLJAXBUtils` ya funciona (y tiene el `marshal` genérico).
3. El `pom.xml` o el módulo del proyecto incluye las dependencias de `jakarta.xml.bind` y lo que ya tenías para JAXB.

Si quieres, después de probar este `main`, me puedes pegar el XML generado y lo comparamos con el del enunciado para afinar detalles (nombres, orden de elementos, etc.).

Tú dijiste:

ah y gracias por todo, eres un crack

ChatGPT Plus

😊 ¡Gracias a ti!

Da gusto ayudarte cuando traes un proyecto tan completo y con ganas de dejarlo fino.

Si quieres:

- revisar el XML generado ✓
- validar la salida con los enunciados ✓
- extender funcionalidades ✓
- añadir DTD/validación automática ✓
- integrar la parte de registro con menú de opciones ✓

...solo me dices y seguimos.

Aquí estoy para que todo quede **redondo y entregable al 100%**.