Cristian Lage Fernández

2.

```
import java.util.Date;

// Clase abstracta Teacher
abstract class Teacher {
    // Atributos
    private String name;
    private String surname;
    private String address;
    private double baseSalary;
    protected double salary; // Accesible para las clases hijas

    // Constructor
    public Teacher(String name, String surname, String address, double baseSalary) {
        this.name = name;
        this.surname = surname;
        this.address = address;
        this.baseSalary = baseSalary;
    }

    // Método abstracto
    public abstract void generatePayroll();

    // Método para enseñar
    public void teach() {
        System.out.println("Dou unha clase de programación en Java");
    }

    // Método principal
```

```
    public static void main(String[] args) {
        // Crear objetos de cada subclase
        CareerOfficer careerOfficer = new CareerOfficer("John", "Doe", "123 Main St", 3000.0, 500.0, 2010, "City Hall");
        Interim interim = new Interim("Jane", "Smith", "456 Oak St", 2500.0, 300.0, "Temporary School");
        Substitute substitute = new Substitute("Bob", "Johnson", "789 Elm St", 2000.0, 100.0, new Date());

        // Establecer valores para los atributos
        careerOfficer.generatePayroll();
        interim.generatePayroll();
        substitute.generatePayroll();

        // Mostrar información y salario de cada profesor
        System.out.println("Chámome " + careerOfficer.getName() + " e o meu salario é de " + careerOfficer.getSalary());
        System.out.println("Chámome " + interim.getName() + " e o meu salario é de " + interim.getSalary());
        System.out.println("Chámome " + substitute.getName() + " e o meu salario é de " + substitute.getSalary());

        // Invocar el método teach() de cada profesor
        careerOfficer.teach();
        interim.teach();
        substitute.teach();
    }
}

// Subclase CareerOfficer
class CareerOfficer extends Teacher {
    // Atributos adicionales
    private double officerComplement;
    private int oppositionYear;
```

```
    private String oppositionPlace;

    // Constructor
    public CareerOfficer(String name, String surname, String address, double baseSalary,
                         double officerComplement, int oppositionYear, String oppositionPlace) {
        super(name, surname, address, baseSalary);
        this.officerComplement = officerComplement;
        this.oppositionYear = oppositionYear;
        this.oppositionPlace = oppositionPlace;
    }

    // Implementación del método abstracto
    public void generatePayroll() {
        salary = baseSalary + officerComplement;
    }

    // Métodos getter para los atributos adicionales
    public double getOfficerComplement() {
        return officerComplement;
    }

    public int getOppositionYear() {
        return oppositionYear;
    }

    public String getOppositionPlace() {
        return oppositionPlace;
    }
}

// Subclase Interim
class Interim extends Teacher {
```

```
    // Atributos adicionales
    private double interimComplement;
    private String destination;

    // Constructor
    public Interim(String name, String surname, String address, double baseSalary,
                   double interimComplement, String destination) {
        super(name, surname, address, baseSalary);
        this.interimComplement = interimComplement;
        this.destination = destination;
    }

    // Implementación del método abstracto
    public void generatePayroll() {
        salary = baseSalary + interimComplement;
    }

    // Métodos getter para los atributos adicionales
    public double getInterimComplement() {
        return interimComplement;
    }

    public String getDestination() {
        return destination;
    }
}

// Subclase Substitute
class Substitute extends Teacher {
    // Atributos adicionales
    private double displacement;
    private Date initDate;
```

```
    // Constructor
    public Substitute(String name, String surname, String address, double baseSalary,
                      double displacement, Date initDate) {
        super(name, surname, address, baseSalary);
        this.displacement = displacement;
        this.initDate = initDate;
    }

    // Implementación del método abstracto
    public void generatePayroll() {
        salary = baseSalary + displacement;
    }

    // Sobrescritura del método teach()
    @Override
    public void teach() {
        System.out.println("Substitúo unha clase de programación en Java");
    }

    // Métodos getter para los atributos adicionales
```

3.

```
import java.util.Date;

// Clase abstracta Teacher
abstract class Teacher {
    // Atributos
    private String name;
    private String surname;
    private String address;
    private double baseSalary;
    protected double salary; // Accesible para las clases hijas
```

```
    // Constructor
    public Teacher(String name, String surname, String address, double baseSalary) {
        this.name = name;
        this.surname = surname;
        this.address = address;
        this.baseSalary = baseSalary;
    }

    // Método abstracto
    public abstract void generatePayroll();

    // Método para enseñar
    public void teach() {
        System.out.println("Dou unha clase de programación en Java");
    }

    // Métodos getter para los atributos
    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }
}

// Subclase CareerOfficer
class CareerOfficer extends Teacher {
    // Atributos adicionales
    private double officerComplement;
    private int oppositionYear;
```

```
    private String oppositionPlace;

    // Constructor
    public CareerOfficer(String name, String surname, String address, double baseSalary,
                         double officerComplement, int oppositionYear, String oppositionPlace) {
        super(name, surname, address, baseSalary);
        this.officerComplement = officerComplement;
        this.oppositionYear = oppositionYear;
        this.oppositionPlace = oppositionPlace;
        generatePayroll(); // Se llama al método de generación de nómina al crear el objeto
    }

    // Implementación del método abstracto
    public void generatePayroll() {
        salary = baseSalary + officerComplement;
    }

    // Métodos getter para los atributos adicionales
    public double getOfficerComplement() {
        return officerComplement;
    }

    public int getOppositionYear() {
        return oppositionYear;
    }

    public String getOppositionPlace() {
        return oppositionPlace;
    }
}

// Subclase Interim
```

```
class Interim extends Teacher {
    // Atributos adicionales
    private double interimComplement;
    private String destination;

    // Constructor
    public Interim(String name, String surname, String address, double baseSalary,
                   double interimComplement, String destination) {
        super(name, surname, address, baseSalary);
        this.interimComplement = interimComplement;
        this.destination = destination;
        generatePayroll(); // Se llama al método de generación de nómina al crear el objeto
    }

    // Implementación del método abstracto
    public void generatePayroll() {
        salary = baseSalary + interimComplement;
    }

    // Métodos getter para los atributos adicionales
    public double getInterimComplement() {
        return interimComplement;
    }

    public String getDestination() {
        return destination;
    }
}

// Subclase Substitute
class Substitute extends Teacher {
    // Atributos adicionales
```

```
    private double displacement;
    private Date initDate;

    // Constructor
    public Substitute(String name, String surname, String address, double baseSalary,
                      double displacement, Date initDate) {
        super(name, surname, address, baseSalary);
        this.displacement = displacement;
        this.initDate = initDate;
        generatePayroll(); // Se llama al método de generación de nómina al crear el objeto
    }

    // Implementación del método abstracto
    public void generatePayroll() {
        salary = baseSalary + displacement;
    }

    // Sobrescritura del método teach()
    @Override
    public void teach() {
        System.out.println("Substitúo unha clase de programación en Java");
    }

    // Métodos getter para los atributos adicionales
    public double getDisplacement() {
        return displacement;
    }

    public Date getInitDate() {
        return initDate;
    }
}
```

```
// Clase principal para el método principal
public class Main {
    public static void main(String[] args) {
        // Crear objetos de cada subclase utilizando los constructores
        CareerOfficer careerOfficer = new CareerOfficer("John", "Doe", "123 Main St", 3000.0, 500.0, 2010, "City Hall");
        Interim interim = new Interim("Jane", "Smith", "456 Oak St", 2500.0, 300.0, "Temporary School");
        Substitute substitute = new Substitute("Bob", "Johnson", "789 Elm St", 2000.0, 100.0, new Date());

        // Mostrar información y salario de cada profesor
        System.out.println("Chámome " + careerOfficer.getName() + " e o meu salario é de " + careerOfficer.getSalary());
        System.out.println("Chámome " + interim.getName() + " e o meu salario é de " + interim.getSalary());
        System.out.println("Chámome " + substitute.getName() + " e o meu salario é de " + substitute.getSalary());

        // Invocar el método teach() de cada profesor
        careerOfficer.teach();
        interim.teach();
        substitute.teach();
    }
}
```

4.

```
import java.util.ArrayList;
import java.util.Date;

// Clase HighSchool
class HighSchool {
    // Atributos
    private String name;
```

```
private ArrayList<Teacher> teachers;

// Constructor
public HighSchool(String name) {
    this.name = name;
    this.teachers = new ArrayList<>();
}

// Método para agregar profesor al ArrayList
public void addTeacher(Teacher teacher) {
    teachers.add(teacher);
}

// Método para encontrar al profesor con el salario más alto
public Teacher mostPaid() {
    Teacher mostPaidTeacher = null;
    double maxSalary = Double.MIN_VALUE;

    for (Teacher teacher : teachers) {
        if (teacher.getSalary() > maxSalary) {
            maxSalary = teacher.getSalary();
            mostPaidTeacher = teacher;
        }
    }

    return mostPaidTeacher;
}

// Método para encontrar al profesor con el salario más bajo
public Teacher leastPaid() {
    Teacher leastPaidTeacher = null;
    double minSalary = Double.MAX_VALUE;
```

```
    for (Teacher teacher : teachers) {
        if (teacher.getSalary() < minSalary) {
            minSalary = teacher.getSalary();
            leastPaidTeacher = teacher;
        }
    }

    return leastPaidTeacher;
}

// Método para calcular la suma de los salarios de todos los profesores
public double salaryCosts() {
    double totalSalary = 0.0;

    for (Teacher teacher : teachers) {
        totalSalary += teacher.getSalary();
    }

    return totalSalary;
}

// Método para calcular la media de los salarios de todos los profesores
public double salaryAverage() {
    if (teachers.isEmpty()) {
        return 0.0;
    }

    return salaryCosts() / teachers.size();
}

// Método principal
```

```
    public static void main(String[] args) {
        // Crear objeto HighSchool
        HighSchool highSchool = new HighSchool("Sample High School");

        // Crear profesores
        CareerOfficer careerOfficer = new CareerOfficer("John", "Doe", "123 Main St", 3000.0, 500.0, 2010, "City Hall");
        Interim interim = new Interim("Jane", "Smith", "456 Oak St", 2500.0, 300.0, "Temporary School");
        Substitute substitute = new Substitute("Bob", "Johnson", "789 Elm St", 2000.0, 100.0, new Date());

        // Agregar profesores al instituto
        highSchool.addTeacher(careerOfficer);
        highSchool.addTeacher(interim);
        highSchool.addTeacher(substitute);

        // Invocar métodos y mostrar resultados
        System.out.println("Profesor con mayor salario: " + highSchool.mostPaid().getName());
        System.out.println("Profesor con menor salario: " + highSchool.leastPaid().getName());
        System.out.println("Costo total de salarios: " + highSchool.salaryCosts());
        System.out.println("Media de salarios: " + highSchool.salaryAverage());
    }
}
```