1.

```
import java.util.ArrayList;

// Clase abstracta Shape (Figura)
abstract class Shape {
    abstract void draw(); // Método abstracto para dibujar
    abstract void erase(); // Método abstracto para borrar
}

// Clase Square (Cadrado) que extiende de Shape
class Square extends Shape {
    @Override
    void draw() {
        System.out.println("Dibujando cadrado");
    }

    @Override
    void erase() {
        System.out.println("Borrando cadrado");
    }
}

// Clase Triangle (Triángulo) que extiende de Shape
class Triangle extends Shape {
    @Override
    void draw() {
        System.out.println("Dibujando triángulo");
    }

    @Override
    void erase() {
        System.out.println("Borrando triángulo");
    }
}

// Clase Circle (Círculo) que extiende de Shape
class Circle extends Shape {
    @Override
    void draw() {
        System.out.println("Dibujando círculo");
    }

    @Override
    void erase() {
        System.out.println("Borrando círculo");
    }
}

// Clase Board (Encerado)
class Board {
    private ArrayList<Shape> shapes = new ArrayList<>(); // ArrayList de figuras
```

```
    // Método para añadir figura al ArrayList
    void addShape(Shape shape) {
        shapes.add(shape);
        shape.draw(); // Llama al método draw() de la figura
    }

    // Método para borrar todas las figuras del ArrayList
    void clearBoard() {
        for (Shape shape : shapes) {
            shape.erase(); // Llama al método erase() de la figura
        }
        shapes.clear(); // Borra todas las figuras del ArrayList
    }
}

// Clase PaintingSession (Sesión de pintura) para probar el funcionamiento
public class PaintingSession {
    public static void main(String[] args) {
        // Crear instancia de Board
        Board board = new Board();

        // Añadir varias figuras al encerado
        board.addShape(new Square());
        board.addShape(new Triangle());
        board.addShape(new Circle());

        // Borrar todas las figuras del encerado
        board.clearBoard();
    }
}
```

2.

```
import java.awt.Color;

// Interfaz Coloreable
interface Coloreable {
    void setColor(Color c); // Método para cambiar el color del objeto
    Color getColor(); // Método para obtener el color del objeto
}

// Clase abstracta Shape (Figura) que implementa la interfaz Coloreable
abstract class Shape implements Coloreable {
    private Color color; // Atributo para almacenar el color de la figura

    // Constructor de Shape
    public Shape() {
        this.color = Color.BLACK; // Color por defecto: negro
    }

    // Implementación de los métodos de la interfaz Coloreable
    @Override
    public void setColor(Color c) {
```

```
        this.color = c;
    }

    @Override
    public Color getColor() {
        return this.color;
    }

    // Métodos abstractos de la clase Shape
    abstract void draw(); // Método abstracto para dibujar
    abstract void erase(); // Método abstracto para borrar
}

// Resto del código sigue igual...

// Clase Square (Cadrado) que extiende de Shape
class Square extends Shape {
    // Implementación de los métodos abstractos
    @Override
    void draw() {
        System.out.println("Dibujando cadrado de color " + getColor());
    }

    @Override
    void erase() {
        System.out.println("Borrando cadrado de color " + getColor());
    }
}

// Clase Triangle (Triángulo) que extiende de Shape
class Triangle extends Shape {
    // Implementación de los métodos abstractos
    @Override
    void draw() {
        System.out.println("Dibujando triángulo de color " + getColor());
    }

    @Override
    void erase() {
        System.out.println("Borrando triángulo de color " + getColor());
    }
}

// Clase Circle (Círculo) que extiende de Shape
class Circle extends Shape {
    // Implementación de los métodos abstractos
    @Override
    void draw() {
        System.out.println("Dibujando círculo de color " + getColor());
    }

    @Override
```

```
    void erase() {
        System.out.println("Borrando círculo de color " + getColor());
    }
}

// Resto del código sigue igual…

3.
// Interfaz Speaker
interface Speaker {
    void speak();
}

// Clase Person
class Person {
    protected String name;
    protected int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

// Clase Student que hereda de Person e implementa Speaker
class Student extends Person implements Speaker {
    private String career;
    private int course;

    public Student(String name, int age, String career, int course) {
        super(name, age);
        this.career = career;
        this.course = course;
    }

    @Override
    public void speak() {
        System.out.println("Ola, son un ALUMNO e sei falar.");
        System.out.println("   Nome: " + name + "   Idade: " + age);
        System.out.println("   Carreira: " + career + "    Curso: " + course);
    }
}

// Clase Teacher que hereda de Person e implementa Speaker
class Teacher extends Person implements Speaker {
    private String office;
    private String email;

    public Teacher(String name, int age, String office, String email) {
        super(name, age);
        this.office = office;
        this.email = email;
```

```
    }

    @Override
    public void speak() {
        System.out.println("Ola, son un PROFESOR e sei falar.");
        System.out.println("   Nome: " + name + "    Idade: " + age);
        System.out.println("   Despacho: " + office + " Email: " + email);
    }
}

// Clase Concierge que hereda de Person e implementa Speaker
class Concierge extends Person implements Speaker {
    private String turn;
    private int seniority;

    public Concierge(String name, int age, String turn, int seniority) {
        super(name, age);
        this.turn = turn;
        this.seniority = seniority;
    }

    @Override
    public void speak() {
        System.out.println("Ola, son un BEDEL e sei falar.");
        System.out.println("   Nome: " + name + "    Idade: " + age);
        System.out.println("   Turno: " + turn + "    Antiguidade: " + seniority);
    }
}

// Clase Device que implementa Speaker
class Device implements Speaker {
    protected int consumption;
    protected int price;

    public Device(int consumption, int price) {
        this.consumption = consumption;
        this.price = price;
    }

    @Override
    public void speak() {
        System.out.println("Ola, son unha " + this.getClass().getSimpleName() + " e sei falar.");
        System.out.println("   Consumo: " + consumption + "    Prezo: " + price);
    }
}

// Clase TV que hereda de Device
class TV extends Device {
    private boolean teletext;
    private int inches;

    public TV(int consumption, int price, boolean teletext, int inches) {
```

```
        super(consumption, price);
        this.teletext = teletext;
        this.inches = inches;
    }

    @Override
    public void speak() {
        super.speak();
        System.out.println("   Teletexto: " + (teletext ? "Si" : "No") + "   Polgadas: " + inches);
    }
}

// Clase Radio que hereda de Device
class Radio extends Device {
    private boolean casette;
    private int power;

    public Radio(int consumption, int price, boolean casette, int power) {
        super(consumption, price);
        this.casette = casette;
        this.power = power;
    }

    @Override
    public void speak() {
        super.speak();
        System.out.println("   Casette: " + (casette ? "Si" : "No") + "   Potencia: " + power);
    }
}

// Programa principal
public class Main {
    public static void main(String[] args) {
        // Crear un array de 7 posiciones para objetos con capacidad de hablar
        Speaker[] speakers = new Speaker[7];

        // Crear objetos y asignar valores a los atributos
        speakers[0] = new Parrot("Macho", 2, "Europa", "Azul");
        speakers[1] = new Tweety("Macho", 6, "Na ducha", 10);
        speakers[2] = new Student("Marta", 22, "Informatica", 3);
        speakers[3] = new Teacher("Jesus", 35, "555-D", "txus@iesteis.es");
        speakers[4] = new Concierge("Dani", 40, "Tarde", 10);
        speakers[5] = new TV(100, 30000, true, 28);
        speakers[6] = new Radio(50, 15000, false, 2);

        // Recorrer el array y llamar al método "speak()" sobre cada objeto
        for (Speaker speaker : speakers) {
            speaker.speak();
            System.out.println(); // Separador entre mensajes
        }
    }
}
```

```
// Clase Parrot que hereda de Bird y implementa Speaker
class Parrot extends Bird implements Speaker {
    private String region;
    private String color;

    public Parrot(String sex, int age, String region, String color) {
        super(sex, age);
        this.region = region;
        this.color = color;
    }

    @Override
    public void speak() {
        System.out.println("Ola, son un LORO e sei falar.");
        System.out.println("   Sexo: " + sex + "     Idade: " + age);
        System.out.println("   Rexion: " + region + "  Cor: " + color);
    }
}

// Clase Tweety que hereda de Bird y implementa Speaker
class Tweety extends Bird implements Speaker {
    private String sing;
    private int numberOfFilms;

    public Tweety(String sex, int age, String sing, int numberOfFilms) {
        super(sex, age);
        this.sing = sing;
        this.numberOfFilms = numberOfFilms;
    }

    @Override
    public void speak() {
        System.out.println("Ola, son PIOLIN e sei falar.");
        System.out.println("   Sexo: " + sex + "     Idade: " + age);
        System.out.println("   Canta: " + sing + "      Peliculas: " + numberOfFilms);
    }
}

// Clase Bird que implementa Speaker
class Bird implements Speaker {
    protected String sex;
    protected int age;

    public Bird(String sex, int age) {
        this.sex = sex;
        this.age = age;
    }

    @Override
    public void speak() {
```

```
        // Método por defecto de la interfaz, se puede dejar vacío o personalizar según
necesidades
        System.out.println("Ola, son un PAJARO e sei falar.");
    }
}
```