1. Стратегия (Strategy)

```
interface Strategy {
   void execute();
}

class ConcreteStrategyA implements Strategy {
   public void execute() {
      System.out.println("Выполнение стратегии A");
   }
}

class Context {
   private Strategy strategy;

   public Context(Strategy strategy) {
      this.strategy = strategy;
   }

   public void executeStrategy() {
      strategy.execute();
   }
}

public class StrategyPattern {
   public static void main(String[] args) {
      Context context = new Context(new ConcreteStrategyA());
      context.executeStrategy();
   }
}
```

2. Шаблонный метод (Template Method)

```
abstract class Game {
   abstract void initialize();
   abstract void startPlay();
   abstract void endPlay();

   public final void play() {
      initialize();
      startPlay();
      endPlay();
   }
}

class Football extends Game {
   void initialize() { System.out.println("Инициализация игры в футбол."); }
   void startPlay() { System.out.println("Игра началась."); }
   void endPlay() { System.out.println("Игра закончена."); }
}

public class TemplateMethodPattern {
   public static void main(String[] args) {
      Game game = new Football();
      game.play();
   }
}
```

3. Команда (Command)

```
interface Command {
   void execute();
}

class Light {
   public void turnOn() {
      System.out.println("Свет включен");
   }
}

class TurnOnLightCommand implements Command {
   private Light light;

   public TurnOnLightCommand(Light light) {
      this.light = light;
   }

   public void execute() {
      light.turnOn();
   }
}

public class CommandPattern {
   public static void main(String[] args) {
      Light light = new Light();
      Command turnOn = new TurnOnLightCommand(light);
      turnOn.execute();
   }
}
```

4. Простая фабрика (Simple Factory)

```
class Pizza {
    public void prepare() {
        System.out.println("Подготовка пиццы");
    }
}

class SimplePizzaFactory {
    public Pizza createPizza() {
        return new Pizza();
    }
}

public class SimpleFactoryPattern {
    public static void main(String[] args) {
        SimplePizzaFactory factory = new SimplePizzaFactory();
        Pizza pizza = factory.createPizza();
        pizza.prepare();
    }
}
```

5. Абстрактная фабрика (Abstract Factory)

```
interface Chair {
   void sitOn();
}

class ModernChair implements Chair {
   public void sitOn() {
      System.out.println("Сидим на современной мебели.");
   }
}

interface FurnitureFactory {
   Chair createChair();
}

class ModernFurnitureFactory implements FurnitureFactory {
   public Chair createChair() {
      return new ModernChair();
   }
}

public class AbstractFactoryPattern {
   public static void main(String[] args) {
      FurnitureFactory factory = new ModernFurnitureFactory();
      Chair chair = factory.createChair();
      chair.sitOn();
   }
}
```

6. Адаптер (Adapter)

```
interface MediaPlayer {
  void play(String audioType);
}

class AdvancedMediaPlayer {
  public void playMp3() {
    System.out.println("Проигрывание MP3");
  }
}

class MediaAdapter implements MediaPlayer {
  private AdvancedMediaPlayer advancedPlayer;

  public MediaAdapter(AdvancedMediaPlayer advancedPlayer) {
    this.advancedPlayer = advancedPlayer;
  }

  public void play(String audioType) {
    if (audioType.equalsIgnoreCase("mp3")) {
      advancedPlayer.playMp3();
    }
  }
}

public class AdapterPattern {
  public static void main(String[] args) {
    MediaPlayer player = new MediaAdapter(new AdvancedMediaPlayer());
    player.play("mp3");
  }
}
```

7. Декоратор (Decorator)

```
interface Coffee {
   String getDescription();
}

class SimpleCoffee implements Coffee {
   public String getDescription() {
      return "Простой кофе";
   }
}

class MilkDecorator implements Coffee {
   private Coffee coffee;

   public MilkDecorator(Coffee coffee) {
      this.coffee = coffee;
   }

   public String getDescription() {
      return coffee.getDescription() + ", с молоком";
   }
}

public class DecoratorPattern {
   public static void main(String[] args) {
      Coffee coffee = new SimpleCoffee();
      Coffee milkCoffee = new MilkDecorator(coffee);
      System.out.println(milkCoffee.getDescription());
   }
}
```

8. Наблюдатель (Observer)

```
import java.util.ArrayList;
import java.util.List;
interface Observer {
  void update();
}
class Subject {
  private List<Observer> observers = new ArrayList<>();
  public void addObserver(Observer observer) {
    observers.add(observer);
  }
  public void notifyObservers() {
    for (Observer observer : observers) {
      observer.update();
    }
  }
}
class ConcreteObserver implements Observer {
  public void update() {
    System.out.println("Наблюдатель уведомлен.");
  }
}

public class ObserverPattern {
  public static void main(String[] args) {
    Subject subject = new Subject();
    Observer observer = new ConcreteObserver();
    subject.addObserver(observer);
    subject.notifyObservers();
  }
}
```

9. Синглтон (Singleton)

```
class Singleton {
    private static Singleton instance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}

public class SingletonPattern {
    public static void main(String[] args) {
        Singleton singleton = Singleton.getInstance();
        System.out.println("Получен экземпляр Singleton: " + singleton);
    }
}
```

10. Итератор (Iterator)

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class IteratorPattern {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");
        Iterator<String> iterator = names.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

11. Фасад (Facade)

```
class Computer {
    public void start() {
        System.out.println("Компьютер включен.");
    }
}

class ComputerFacade {
    private Computer computer;

    public ComputerFacade() {
        this.computer = new Computer();
    }

    public void startComputer() {
        computer.start();
    }
}
public class FacadePattern {
    public static void main(String[] args) {
        ComputerFacade facade = new ComputerFacade();
        facade.startComputer();
    }
}
```