


SPRAWOZDANIE NR			
Nazwa ćwiczenia	Polimorfizm		 <b>POLITECHNIKA BYDGOSKA</b> Wydział Telekomunikacji, Informatyki i Elektrotechniki
Przedmiot	Programowanie obiektowe		
Student grupa	Paweł Jońca gr 7		
Data ćwiczeń	02.12	08.12	Data oddania sprawozdania

Klasa abstrakcyjna służy nam jako baza dla innych klas. Pozwala nam uniknąć powtarzania kodu w klasach dziedziczących. Pozwala nam tworzyć logiczną hierarchię klas i zapewnia wspólną strukturę dla obiektów o podobnych cechach.

W moim kodzie: 'Vehicle' to klasa abstrakcyjna, która reprezentuje ogólny typ pojazdu

Zawiera metodę abstrakcyjną 'drive', która nie ma implementacji w klasie bazowej. Wymusza na klasach dziedziczących dostarczenie własnej impleme

```

1 // Klasa abstrakcyjna i polimorfizm
2 abstract class Vehicle { 5 usages 3 inheritors
3     // Metoda abstrakcyjna
4     public abstract void drive(); 1 usage 3 implementations
5 }

```

Klasy dziedziczące automatycznie przejmują wszystkie publiczne i chronione pola oraz metody z klasy bazowej. Klasy dziedziczące mogą dodawać swoje własne metody i pola, aby zapewnić dodatkową funkcjonalność. Mogą również zmieniać zachowanie metod klasy bazowej przez ich przesłanianie

Car dziedziczy po klasie Vehicle i implementuje metodę drive

Tak samo Bike oraz Boat

```
// Klasa Car dziedziczy po klasie Vehicle
class Car extends Vehicle { 1 usage
    @Override 1 usage
    public void drive() {
        System.out.println("Samochód jedzie po drodze.");
    }
}

// Klasa Bike dziedziczy po klasie Vehicle
class Bike extends Vehicle { 1 usage
    @Override 1 usage
    public void drive() {
        System.out.println("Rower jedzie po ścieżce rowerowej.");
    }
}

// Klasa Boat dziedziczy po klasie Vehicle
class Boat extends Vehicle { 1 usage
    @Override 1 usage
    public void drive() {
        System.out.println("Łódź płynie po wodzie.");
    }
}
```

## Definicja interfejsu

Refuelable reprezentuje obiekty, które mogą zostać zatankowane lub naładowane

Zawiera metodę refuel, która muszą zaimplementować klasy implementujące interfejs

Interfejs umożliwia grupowanie klas według wspólnej cechy ('refuel') niezależnie od ich dziedziczenia

```
// Interfejs i polimorfizm
interface Refuelable { 5 usages 3 implementations
    void refuel(); 1 usage 3 implementations
}
```

## Definicja GasStation

Implementuje metodę refuel dla stacji benzynowej

```
// Klasa implementująca interfejs Refuelable
class GasStation implements Refuelable { 1 usage
    @Override 1 usage
    public void refuel() {
        System.out.println("Pojazd tankuje paliwo na stacji benzynowej.");
    }
}
```

```
// Klasa implementująca interfejs Refuelable
class ChargingStation implements Refuelable { 1 usage
    @Override 1 usage
    public void refuel() {
        System.out.println("Pojazd elektryczny ładuje baterię na stacji ładowania.");
    }
}

// Klasa implementująca interfejs Refuelable
class Marina implements Refuelable { 1 usage
    @Override 1 usage
    public void refuel() {
        System.out.println("Łódź tankuje paliwo w marinie.");
    }
}
```

## Tworzenie tablicy vehicles

```
public class Main {  
    public static void main(String[] args) {  
        // Sekcja A: Klasa abstrakcyjna i polimorfizm  
        System.out.println("=== Klasa abstrakcyjna ===");  
        Vehicle[] vehicles = {new Car(), new Bike(), new Boat()}; // Rzutowanie w górę do klasy bazowej  
  
        // Wywołania polimorficzne metody drive  
        for (Vehicle vehicle : vehicles) {  
            vehicle.drive(); // Każda klasa potomna wywołuje własną implementację  
        }  
        // Sekcja B: Interfejs i polimorfizm  
        System.out.println("\n--- Interfejs ---");  
        Refuelable[] stations = {new GasStation(), new ChargingStation(), new Marina()}; // Rzutowanie do interfejsu  
  
        // Wywołania polimorficzne metody refuel  
        for (Refuelable station : stations) {  
            station.refuel(); // Każda klasa wywołuje własną implementację  
        }  
    }  
}
```

```
=== Klasa abstrakcyjna ===  
Samochód jedzie po drodze.  
Rower jedzie po ścieżce rowerowej.  
Łódź płynie po wodzie.  
  
--- Interfejs ---  
Pojazd tankuje paliwo na stacji benzynowej.  
Pojazd elektryczny ładuje baterię na stacji ładowania.  
Łódź tankuje paliwo w marinie.
```

## Wnioski :

Wykonując to zadanie poznałem jak bardzo ważny jest polimorfizm w programowaniu. Dzięki wykorzystaniu klasy abstrakcyjnej i interfejsu kod staje się bardziej elastyczny i łatwiejszy do rozbudowy, bo można dodawać nowe funkcjonalności bez ingerencji w istniejące elementy. Polimorfizm pozwala obsługiwać różne obiekty w jednolity sposób, co upraszcza pisanie i organizację programu. Taki podział na klasy i interfejsy sprawia, że kod jest przejrzysty i łatwiejszy do zrozumienia.

```

// Klasa abstrakcyjna i polimorfizm
abstract class Vehicle {
    // Metoda abstrakcyjna
    public abstract void drive();
}

// Klasa Car dziedziczy po klasie Vehicle
class Car extends Vehicle {
    @Override
    public void drive() {
        System.out.println("Samochód jedzie po drodze.");
    }
}

// Klasa Bike dziedziczy po klasie Vehicle
class Bike extends Vehicle {
    @Override
    public void drive() {
        System.out.println("Rower jedzie po ścieżce rowerowej.");
    }
}

// Klasa Boat dziedziczy po klasie Vehicle
class Boat extends Vehicle {
    @Override
    public void drive() {
        System.out.println("Łódź płynie po wodzie.");
    }
}

// Interfejs i polimorfizm
interface Refuelable {
    void refuel();
}

// Klasa implementująca interfejs Refuelable
class GasStation implements Refuelable {
    @Override
    public void refuel() {
        System.out.println("Pojazd tankuje paliwo na stacji benzynowej.");
    }
}

// Klasa implementująca interfejs Refuelable
class ChargingStation implements Refuelable {
    @Override
    public void refuel() {
        System.out.println("Pojazd elektryczny ładuje baterię na stacji ładowania.");
    }
}

// Klasa implementująca interfejs Refuelable
class Marina implements Refuelable {
    @Override
    public void refuel() {
        System.out.println("Łódź tankuje paliwo w marinie.");
    }
}

public class Main {
    public static void main(String[] args) {
        // Sekcja A: Klasa abstrakcyjna i polimorfizm
        System.out.println("=== Klasa abstrakcyjna ===");
        Vehicle[] vehicles = {new Car(), new Bike(), new Boat()}; //
Rzutowanie w górę do klasy bazowej

```

```

        // Wywołania polimorficzne metody drive
        for (Vehicle vehicle : vehicles) {
            vehicle.drive(); // Każda klasa potomna wywołuje własną
implementację
        }
        // Sekcja B: Interfejs i polimorfizm
        System.out.println("\n--- Interfejs ---");
        Refuelable[] stations = {new GasStation(), new ChargingStation(),
new Marina()}; // Rzutowanie do interfejsu

        // Wywołania polimorficzne metody refuel
        for (Refuelable station : stations) {
            station.refuel(); // Każda klasa wywołuje własną implementację
        }
    }
}

```