Java: triedy a rozhrania

Základné konštrukcie jednoducho

Polia, metódy, konštruktory, modifikátory prístupu, rozhrania, abstraktné triedy, record a enum

1. Z čoho sa skladá trieda?

Trieda môže obsahovať:

- polia — údaje objektu;
- konštruktory — vytváranie objektov;
- metódy — činnosti objektu.

Jednoduchý príklad:

```
public class Student {

    private String name;
    private int age;

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

    public void introduce() {
        System.out.println(
            "My name is " + name
        );
    }
}
```

Tu:

- `name` a `age` sú polia;
- `Student(...)` je konštruktor;
- `introduce()` je metóda.

Hlavná myšlienka

Trieda opisuje, aké údaje objekt uchováva a aké činnosti dokáže vykonávať.

2. Vytvorenie objektu

Objekt sa vytvára pomocou kľúčového slova `new`:

```
Student student =
    new Student("Anna", 14);
```

Tu:

```
Student
```

je typ premennej,

```
student
```

je názov premennej,
a

```
new Student("Anna", 14)
```

vytvorí nový objekt.
Potom môžeme zavolať jeho metódu:

```
student.introduce();
```

3. Konštruktor

Konštruktor sa volá pri vytváraní nového objektu.
Jeho názov je rovnaký ako názov triedy:

```
public class Book {

    private String title;

    public Book(String title) {
        this.title = title;
    }
}
```

Konštruktor nemá návratový typ.
Preto nepíšeme:

```
// Nespravne
public void Book(String title) {
}
```

Takýto kód už deklaruje obyčajnú metódu, nie konštruktor.
Kľúčové slovo `this` označuje **aktuálny objekt**:

```
this.title = title;
```

Vľavo je pole objektu a vpravo parameter konštruktora.

4. Viacero konštruktorov

Trieda môže mať viacero konštruktorov s rôznymi parametrami.

```
public class User {

    private String name;
    private boolean active;

    public User(String name) {
        this(name, true);
    }

    public User(
            String name,
            boolean active) {

        this.name = name;
        this.active = active;
    }
}
```

Volanie

```
this(name, true);
```

volá iný konštruktor tej istej triedy.
Takéto volanie musí byť prvým príkazom v konštruktore.

5. Modifikátory prístupu

Modifikátory prístupu určujú, odkiaľ možno pristupovať k poliam a metódam.

	Vlastná trieda	Balík	Podtrieda	Všade
`private`	áno	nie	nie	nie
`bez modifikátora`	áno	áno	iba v tom istom balíku	nie
`protected`	áno	áno	áno	nie
`public`	áno	áno	áno	áno

Na začiatok je užitočné zapamätať si dve hlavné možnosti:
private
sa používa pre vnútorný stav triedy.
public
sa používa pre operácie, ktoré majú byť dostupné inému kódu.
Napríklad:

```
public class Player {

    private int health;

    public int getHealth() {
        return health;
    }
}
```

6. static

Kľúčové slovo **static** znamená, že pole alebo metóda patrí samotnej triede, nie konkrétnemu objektu.
Napríklad:

```
public class MathHelper {

    public static int square(int x) {
        return x * x;
    }
}
```

Metódu môžeme zavolať bez vytvorenia objektu:

```
int result = MathHelper.square(5);
```

Výsledok:

```
25
```

Dôležité

Bežná metóda sa volá na objekte.

```
student.introduce();
```

Statická metóda sa zvyčajne volá cez názov triedy.

```
MathHelper.square(5);
```

7. final

Kľúčové slovo **final** má viacero významov podľa toho, kde sa používa.

final pole

Po priradení nemožno hodnotu alebo referenciu nahradiť inou:

```
private final String name;
```

final metóda

Nemožno ju prepísať v podtriede:

```
public final void print() {
    System.out.println("Hello");
}
```

final trieda

Nemožno z nej dediť:

```
public final class Calculator {
}
```

Pre konštanty sa často používajú `static` a `final` spolu:

```
public static final int MAX_SIZE = 100;
```

8. Čo je rozhranie?

Rozhranie určuje kontrakt:

„čo musí objekt vedieť robiť“.

Napríklad:

```
public interface Printable {

    void print();
}
```

Rozhranie hovorí, že objekt musí poskytovať metódu `print()`.
Trieda môže toto rozhranie implementovať:

```
public class Document
        implements Printable {

    @Override
    public void print() {
        System.out.println(
            "Printing document"
```

```
        );
    }
}
```

Kľúčové slovo

```
implements
```

znamená, že trieda implementuje rozhranie.

9. Prečo potrebujeme rozhrania?

Rozhranie umožňuje pracovať s rôznymi objektmi rovnakým spôsobom.
Napríklad:

```
public class Photo
        implements Printable {

    @Override
    public void print() {
        System.out.println(
            "Printing photo"
        );
    }
}
```

Teraz môžeme `Document` aj `Photo` používať cez spoločný typ:

```
Printable item = new Photo();

item.print();
```

Môžeme vytvoriť aj pole rôznych objektov:

```
Printable[] items = {
    new Document(),
    new Photo()
};

for (Printable item : items) {
    item.print();
}
```

Preto rozhrania dobre spolupracujú s polymorfizmom.

10. Trieda môže implementovať viacero rozhraní

Java neumožňuje, aby jedna trieda dedila naraz z viacerých tried.
Jedna trieda však môže implementovať viacero rozhraní.
Napríklad:

```
public interface Printable {
    void print();
}

public interface Savable {
    void save();
}
```

Jedna trieda môže implementovať obe:

```
public class Document
        implements Printable, Savable {

    @Override
    public void print() {
        System.out.println("Print");
    }

    @Override
    public void save() {
        System.out.println("Save");
    }
}
```

Objekt `Document` má teraz dve úlohy:

- možno ho vytlačiť;
- možno ho uložiť.

11. Abstraktná trieda

Abstraktná trieda je trieda, ktorej objekt nemožno vytvoriť priamo.
Môže obsahovať:

- polia;
- konštruktory;
- bežné metódy;
- abstraktné metódy.

Napríklad:

```
public abstract class Animal {

    private final String name;

    public Animal(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public abstract void sound();
}
```

Metóda

```
public abstract void sound();
```

nemá implementáciu.
Musí ju poskytnúť podtrieda:

```
public class Dog extends Animal {

    public Dog(String name) {
        super(name);
    }
```

```
    @Override
    public void sound() {
        System.out.println("Woof!");
    }
}
```

Volanie

```
super(name);
```

volá konštruktor rodičovskej triedy.

12. Abstraktná trieda alebo rozhranie?

Zjednodušené pravidlo:
Abstraktná trieda
je vhodná, keď príbuzné triedy zdieľajú spoločné údaje a spoločnú implementáciu.
Napríklad:

Dog is an Animal.

Rozhranie
je vhodné, keď chceme opísať schopnosť alebo úlohu objektu.
Napríklad:

Document is Printable.

	Abstraktná trieda	Rozhranie
Polia objektu	áno	zvyčajne nie
Konštruktor	áno	nie
Hotové metódy	áno	áno, cez `default`
Koľko možno použiť	dedí sa jedna trieda	možno implementovať viacero
Hlavná myšlienka	spoločný základ	spoločný kontrakt

13. default metódy v rozhraní

Rozhranie môže obsahovať metódu s hotovou implementáciou.
Používa sa na to slovo `default`:

```
public interface Greeting {

    void sayHello();

    default void sayBye() {
        System.out.println("Bye!");
    }
}
```

Trieda musí implementovať:

```
sayHello()
```

ale už dostane hotovú metódu:

```
sayBye()
```

Dôležité

Na prvé zoznámenie stačí pamätať:
bežná metóda rozhrania určuje povinný kontrakt, zatiaľ čo `default` metóda už obsahuje hotovú implementáciu.

14. record

record je vhodný pre malé objekty, ktorých hlavnou úlohou je uchovávať údaje. Napríklad:

```
public record Student(
    long id,
    String name
) {
}
```

Vytvorenie objektu:

```
Student student =
    new Student(1, "Anna");
```

Získanie údajov:

```
long id = student.id();
String name = student.name();
```

Java automaticky vytvorí pre `record`:

- konštruktor;
- prístupové metódy;
- `equals()`;
- `hashCode()`;
- `toString()`.

Dôležité

Bežná trieda je vhodná pre objekty so správaním a podľa potreby aj s meniteľným stavom.
`record` je obzvlášť vhodný pre jednoduché hodnotové objekty.

15. enum

enum sa používa vtedy, keď existuje vopred známy a obmedzený počet možností. Napríklad dni v týždni:

```
public enum Day {
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
}
```

Alebo stav objednávky:

```
public enum OrderStatus {
    NEW,
    PAID,
    SHIPPED,
    CANCELLED
}
```

Použitie:

```
OrderStatus status =
    OrderStatus.PAID;
```

Výhodou `enum` je, že hodnota môže byť iba jednou z povolených možností.

16. Generics: typ v špicatých zátvorkách

Generics umožňujú triede alebo rozhraniu pracovať s rôznymi typmi údajov.
Už sme ich videli pri kolekciách:

```
List<String>
List<Integer>
```

Môžeme vytvoriť aj vlastnú generickú triedu:

```
public class Box<T> {

    private final T value;

    public Box(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
}
```

Teraz môžeme tú istú triedu používať s rôznymi typmi:

```
Box<String> textBox =
    new Box<>("Hello");

Box<Integer> numberBox =
    new Box<>(42);
```

Písmeno `T` predstavuje typ, ktorý sa určí neskôr.

Dôležité

Generics umožňujú napísať jednu všeobecnú triedu a pritom zachovať kontrolu typov kompilátorom.

17. sealed typy

Ide o modernejšiu možnosť v Jave.
Kľúčové slovo **sealed** umožňuje presne obmedziť, ktoré triedy môžu implementovať

rozhranie alebo dediť z triedy.
Napríklad:

```
public sealed interface Result
        permits Success, Failure {
}
```

Povolené sú iba uvedené typy:

```
public final class Success
        implements Result {
}

public final class Failure
        implements Result {
}
```

Je to užitočné vtedy, keď je množina možných typov vopred známa a obmedzená.
Napríklad:

$$\text{Result} \longrightarrow \begin{cases} \text{Success}, \\ \text{Failure}. \end{cases}$$

Pre prvé programy v Jave nie je `sealed` nevyhnutná téma, ale je užitočné vedieť, že táto možnosť v modernej Jave existuje.

18. equals(), hashCode() a toString()

Všetky triedy v Jave dedia základné metódy z triedy `Object`.
Patria medzi ne:
equals()
porovnáva objekty.
hashCode()
vracia hašovací kód objektu.
toString()
vytvára textovú reprezentáciu objektu.
Napríklad:

```
System.out.println(student);
```

Pre `record` sa tieto metódy automaticky vytvoria podľa jeho komponentov.
Pri bežných triedach ich niekedy treba prepísať manuálne.

19. Časté chyby

Chyba 1.
Nastaviť všetky polia ako `public`:

```
public int health;
```

Stav objektu je zvyčajne lepšie chrániť pomocou `private`.

Chyba 2.
Pomýliť si konštruktor s metódou.
Konštruktor:

```
public Student(String name) {
}
```

Metóda:

```
public void print() {
}
```

Chyba 3.
Pokúsiť sa vytvoriť objekt abstraktnej triedy:

```
// Nie je dovolene
Animal animal = new Animal("Bob");
```

Chyba 4.
Uviesť, že trieda implementuje rozhranie, ale zabudnúť implementovať jeho povinnú metódu.

Chyba 5.
Použiť dedičnosť iba preto, aby sme znovu využili existujúci kód.
Dedičnosť by mala vyjadrovať skutočný vzťah medzi typmi.

20. Stručný prehľad

class
Opisuje stav a správanie objektov.
constructor
Vytvára a pripravuje objekt.
private
Skrýva časť triedy pred vonkajším kódom.
public
Sprístupňuje triedu, pole alebo metódu inému kódu.
static
Patrí triede, nie konkrétnemu objektu.
final
Obmedzuje ďalšie priradenie, prepisovanie alebo dedenie.
interface
Určuje spoločný kontrakt alebo úlohu.
abstract class
Poskytuje spoločný základ pre príbuzné triedy.
record
Pohodlne uchováva skupinu súvisiacich hodnôt.
enum
Opisuje pevný súbor možných hodnôt.
generics
Umožňujú tomu istému kódu bezpečne pracovať s rôznymi typmi.

Hlavný princíp

Použite **triedu**, keď objekt potrebuje uchovávať stav a vykonávať činnosti.
Použite **rozhranie**, keď potrebujete opísať spoločnú schopnosť alebo úlohu:

„čo objekt dokáže robiť“.

Použite **abstraktnú triedu**, keď majú príbuzné triedy spoločný základ.
Použite **record** pre jednoduché hodnotové objekty.
Použite **enum**, keď je množina povolených možností vopred známa.
Hlavnou myšlienkou nie je používať jazykovú konštrukciu iba preto, že je zložitejšia

alebo modernejšia.
Vyberte tú, ktorá danú úlohu opisuje najjasnejšie.