Dědičnost

Dědičnost je jedna ze základních vlastností objektově orientovaného programování. Umožňuje vytvořit hierarchii objektů, které mají podobné vlastnosti, a to pak může simulovat realitu a reálné objekty, bez nutnosti zbytečně přepisovat veškeré atributy tříd. Ušetříme si tak tedy duplicitní kód. Třída může dědit jinou třídu pomocí klíčového slova **extends**.

Na rozdíl od ostatních programovacích jazyků Java nenabízí vícenásobnou dědičnost, znamená to tedy, že jedna třída může vždy mít pouze jednu rodičovskou třídu, třídu kterou dědí.

Pokud chceme, aby potomek rodičovské třídy měl přístup k atributům nebo metodám musíme jim přiřadit modifikátor přístupu **protected**. Ten zajistí přístup všem potomkům v jakémkoliv balíčku, nebo jakékoliv třídě balíčku stejného. Pokud však nastavíme modifikátor přístupu jako **private** bude atribut nebo metoda přístupná pouze třídě, ve které je atribut nebo metoda definována.

Konstruktory v dědičnosti

Pokud třída dědí, bude po ní v konstruktoru požadována inicializace konstruktoru rodičovské třídy, kterou provedeme skrze příkaz **super()**, kterému poskytneme všechny parametry požadované rodičovským konstruktorem.

Konstruktory se takto volají od nejvyšší třídy v hierarchii a postupně se volání posouvá na konstruktory, které jsou na úrovni nižší.

Přepisování metod(polymorfizmus)

V případě dědičnosti můžeme narazit na případ, ve kterém chceme, aby jedna metoda jiné věci. Například každý zaměstnanec má práci, ale každý zaměstnanec má jinou náplň práce. Metoda se bude jmenovat stejně, ale bude dělat něco úplně jiného. V Javě toho docílíme tím, že nad metodu připíšeme anotaci **@Override**, která značí, že metodu přepisujeme. Anotace **@Override** není povinná, ale bez ní, kompilátor nebude kontrolovat jestli opravdu přepisujeme metodu (programátor může špatně napsat název metody a místo přepsaní tak vytvoří novou metodu). Poté můžeme tělo metody upravit dle libosti a obsah se může naprosto lišit od definice v rodičovské třídě.

Abstraktní metody(třídy)

Abstraktní metody jsou metody, které slouží jako prototyp. Abstraktní metoda neobsahuje tělo, ale obsahuje pouze pojmenování, návratový typ a seznam parametrů. Pokud se v třídě nachází abstraktní metoda musí být samotná třída deklarována jako abstraktní. Všechny abstraktní třídy a metody musí být označeny klíčovým slovem **abstract**. Z takovéto třídy nelze vytvořit instanci, ale

```
public abstract class Employee {
   private String name;
   private String address;
   private int number;

   public abstract double computePay();
   // Remainder of class definition
}
```

abstraktní třída

ostatní třídy ji mohou dědit a tyto třídy poté musí definovat všechny těla metod, které abstraktní třída obsahuje.

Klíčové slova public, private, protected a final

Public – označuje atributy nebo metody jako veřejné, a jsou tak zpřístupněny celému programu. Mohou se volat odkudkoliv, v jakémkoliv balíčku či třídě.

Private – uzavírá přístupnost proměnné nebo metody pouze na danou třídu a to i v případě dědění, kdy proměnná nebo metoda s modifikátorem přístupnosti **private** nebude pro potomka viditelná. Pokud chceme, aby proměnná byla přístupná z jiné třídy, musíme vytvořit public metody, které mohou s těmito proměnnými pracovat. Nazývají se gettry(získávání proměnné) a settry(nastavování proměnné).

Protected – proměnná nebo metoda s takovýmto modifikátorem přístupnosti bude viditelná pro všechny potomky tříd, které v sobě mají definované atributy s klíčovým slovem **protected** a to v jakémkoliv balíčku, a nebo pro všechny třídy, ale pouze ve stejném balíčku.

Není definováno – pokud není definována proměnná jako public, private nebo protected tak se chová jako public, ale je viditelná pouze uvnitř svého balíčku.

Final – Pokud je s klíčovým slovem definována proměnná, znamená to, že po jejím inicializování již její hodnota nepůjde za žádných podmínek změnit.

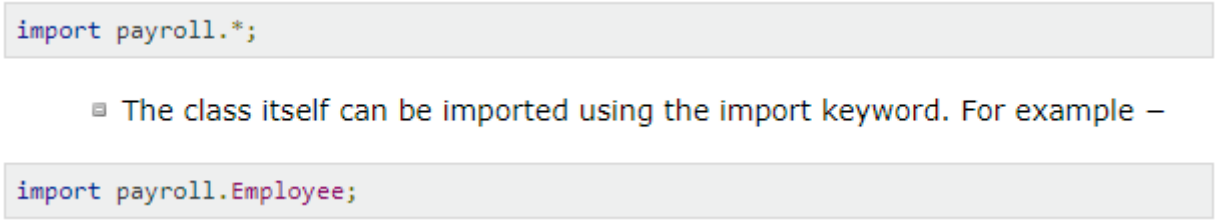
Pokud je s klíčovým slovem definována metoda, znamená to, že potomek tuto metodu nemůže přepisovat (nemůže být využit polymorfismus).

Pokud je s klíčovým slovem definována třída, znamená to, že tato třída nemůže být dále děděna jinými třídami.

Balíčky

Balíčky se v Javě používají proto, aby se předešlo konfliktům při pojmenování, omezení přístupu k proměnným a metodám a rozdělení programů do sekcí například podle funkčnosti, což může v některých případech ulehčit hledání některých tříd nebo rozhraní podle jejich funkčnosti.

Pokud chce třída použít jinou třídu ze stejného balíčku není nutné používat importování. Pokud však chce třída využít jinou třídu, která se nenachází ve stejném balíčku, tak si musí balíček importovat a to buď celý, nebo může importovat pouze jeho část/třídu.



import balíčků/částí balíčků

Rozhraní a implementace

V podstatě se podobá třídě. Jedná se o sbírku abstraktních metod, metody slouží tedy pouze jako prototyp (definováno je pouze jméno, návratový typ a parametry). Výhoda rozhraní oproti abstraktní třídě je ten, že nepodléhá pravidlům dědičnosti a jedna třída tak může implementovat více rozhraní. Třída implementuje rozhraní klíčovým slovem **implements** a poté musí daná třída obsahovat definici všech metod, které se v implementovaném rozhraní nachází. Pokud ve třídě která implementuje rozhraní nejsou definovány všechny metody z rozhraní, musí být třída definována jako abstraktní. Rozhraní se mohou navzájem dědit. U dědičnosti rozhraní neplatí stejná pravidla jako u dědičnosti tříd. To znamená, že jedno rozhraní může dědit více rozhraní a v tomto případě je vícenásobná dědičnost v Javě povolena.

```java
// Filename: Sports.java
public interface Sports {
   public void setHomeTeam(String name);
   public void setVisitingTeam(String name);
}

// Filename: Football.java
public interface Football extends Sports {
   public void homeTeamScored(int points);
   public void visitingTeamScored(int points);
   public void endOfQuarter(int quarter);
}

// Filename: Hockey.java
public interface Hockey extends Sports {
   public void homeGoalScored();
   public void visitingGoalScored();
   public void endOfPeriod(int period);
   public void overtimePeriod(int ot);
}
```

dědičnost rozhraní

```java
public interface Hockey extends Sports, Event
```

mnohonásobná dědičnost rozhraní

Rozhraní – default

Od javy 8 je možno v rozhraní definovat tělo metody pokud metodu definujeme slovem default. Takováto metoda může být definována uvnitř rozhraní a metoda se tak sama implementuje do každé třídy, která implementuje rozhraní. Můžeme tak obejít zbytečnou implementaci metod pokud chceme měnit strukturu rozhraní v již zaběhnuté struktuře programu.