## Dědičnost

Třída, které se říká **rodičovská nebo nadtřída**, definuje své proměnné a metody. Její **podtřída, také potomek nebo zděděná třída**, převezme ty vlastnosti, které se jí hodí, upraví si ty vlastnosti, které se jí nehodí, a přidá své nové vlastnosti, které v původní třídě chybí. Je tak možné využít stávající kód, například otestovaný a dobře odladěný, k vytvoření nových tříd, aniž by ho bylo nutné opakovat.

### Zděděná třída

K vytvoření třídy, která je potomkem, je zapotřebí název rodičovské třídy. Tento název se při deklaraci uvede spolu s klíčovým slovem **extends** za jméno nově vytvářené třídy.

Uvažujme o existenci robotického psa, který je na baterie, ale jinak svých chováním věrně simuluje chování psa živého.

Založme si tedy novou třídu Robot, která bude potomkem rodičovské třídy Pes. Robot tak zdědí všechny atributy a metody rodičovské třídy, může je upravit, případně doplnit o své vlastní.

## Založení třídy

V rámci balíčku s naším projektem si založíme novou třídu. Klikneme pravým tlačítkem na název našeho projektu (Hlavni) a zvolíme volbu New -> Java Class (viz. obrázek).



Vytvoří se nám nový soubor, určený pro novou třídu. Její název stanovíme na Robot.

Pomocí klíčového slova extends vytvoříme potomka třídy Pes:

public class Robot **extends** Pes {

zdědíme tak všechny proměnné a metody rodičovské třídy (pokud jsou označené modifikátorem přístupu public, případně protected). Kromě toho si založíme vlastní proměnné a metody. První proměnnou bude stav baterie, náš robotický pes potřebuje (na rozdíl od toho živého) mít akumulátor a budeme chtít sledovat stav nabití této baterie.

public class Robot extends Pes {

private int kapacitabaterie;

Nyní je potřeba vytvořit konstruktor naší nové třídy, který v sobě **jako první věc** zavolá konstruktor rodičovské třídy. Jelikož naše rodičovská třída Pes má přetížené konstruktory a disponuje hned třemi, je nutné zvážit, který z nich to bude.

O klíčovém slovu this, které obsahuje odkaz na aktuální instance třídy, víme. Podobným způsobem by bylo vhodné mít přístup i k odkazu na rodiče. Ten obsahuje klíčové slovo **super** a můžeme pomocí něho přistupovat k veřejným atributům a metodám rodiče.

Pomocí klíčového slova **super** předáme některé formální parametry konstruktoru našeho předka (barva) a naopak některé použijeme k nastavení atributů naší třídy (kapacitabaterie).

public class Robot extends Pes {

private int kapacitabaterie;

**public Robot(String barva, int kapacitabaterie)** {

super(barva); //na prvním místě voláme konstruktor rodiče, v tomto případě č. 2

this.kapacitabaterie = kapacitabaterie;

}

****V případě, kdy rodičovská třída obsahuje konstruktor s parametry, musí zděděná třída uvést konstruktor s jeho voláním i v případě, že by svůj konstruktor jinak nepotřebovala. Pokud rodičovská třída kromě konstruktoru s parametry obsahuje i bezparametrický konstruktor, který chce potomek použít, nemusí tento konstruktor rodiče volat.

V duchu pravidel zapouzdření máme označenou naší proměnnou kapacitabaterie jako private. Nebude k ní tedy možný přímý přístup odjinud než z naší třídy. Vytvoříme tedy veřejnou metodu pro případnou manipulaci s touto proměnnou:

public int getBaterie(){

return this.kapacitabaterie;

}

Nyní se vrátíme do hlavní funkce main, kde si našeho první robotického psa (tzn. objekt) na základě třídy Robot vytvoříme.

Robot robotPunta = new Robot("ruzova",1500); // zavola svuj konstruktor a také pouzije konstruktor rodiče č. 2

Vytvoříme tak robota Puňtu, který bude mít růžovou barvu (provedení pro dívky) a plný stav jeho baterie při vytvoření bude 1500 jednotek.

Vyzkoušíme, zda náš pes také něco umí a zavoláme si metodu předka štěkej:

robotPunta.stekej(3); //používá metodu rodiče

Tuto metodu jsme ve třídě Robot nedefinovali, přesto jí má náš robotický Puňta k dispozici, protože jí jeho třída Robot zdědila od rodičovské třídy Pes.

Můžeme si také pomocí metody getBaterie vypsat stav baterie našeho psíka.

System.out.println(robotPunta.getBaterie());

Vytvoříme si dále metodu SetBaterie, která bude odčerpávat určený počet jednotek z baterky.

public void setBaterie(int kapacitabaterie) {

this.kapacitabaterie = this.kapacitabaterie - kapacitabaterie;

}

Pokud si chceme vyzkoušet manipulaci s vnitřní proměnnou našeho robotické psíka, může vytvořit jeho vlastní metodu funguj, která při každém zavolání provede nějaký psí kousek a také vyčerpá určený počet jednotek z jeho bateriové kapacity.

Připíšeme do třídy Robot metodu funguj:

public void funguj(int kolikrat) {

for (int i=0; i<kolikrat; i++){

System.out.printf("funguji ");

this.setBaterie(10);

}

}

V metodě main pak zavoláme příslušnou metodu, můžeme doplnit o přání uživatele, kolikrát má pes fungovat.

System.out.println("Kolikrát má robotický pes zafungovat?");

robotPunta.funguj(sc.nextInt()); //používá vlastní metodu. Zafungování stojí robota 10 jednotek z jeho baterie

System.out.println(robotPunta.getBaterie());

Stav ověříme následným vypsáním stavu baterie (viz. Poslední řádek).

**Statická proměnná a metoda třídy Robot**

Na třídě Robot si ještě vyzkoušíme statické proměnné a metody – přidáme si počítadlo robotických psíků.

public class Robot extends Pes {

private **static** int pocetrobotu; //statická proměnná třídy Robot

…

Aby se nám proměnná při každém založení nového robotického psa inkrementovala, přidáme příslušný kód ke konstruktoru třídy Robot.

pocetrobotu = pocetrobotu+1;

Pokud budeme chtít znát stav naší proměnné, založíme si statickou metodu třídy Robot.

public **static** int getPocet() {

return pocetrobotu;

}

V hlavní metodě main si založme ještě dalšího robotického psa, např. robotAzor. Bude mít modrou barvu (provedení pro chlapce) a bude disponovat větší baterií.

Robot robotAzor = new Robot("modra",3000);

Na konec metody main pak zavolejme statickou metodu celé třídy Robot, která nám sdělí, kolik robotických psíků máme založených.

System.out.println(Robot.getPocet());

## Překrývání metod

Nesoukromé metody rodičovské třídy, se kterými jsme spokojeni, můžeme používat tak, jak jsou. Metody, které nám chybí, není problém doplnit. Pak jsou metody, které rodičovská třída obsahuje, **ale nechceme je použít stejným způsobem** i ve zděděné třídě. Řešením tohoto problému je dané metody překrýt. Ve zděděné třídě budou tyto metody uvedeny znovu, se stejnou návratovou hodnotou, stejnými parametry, stejným názvem a stejným identifikátorem přístupu, ale s jinými příkazy.

public void behej(int nactivzdalenost){ //překrytá metoda běhej

for (int i=0; i<nactivzdalenost; i++){

System.out.printf("robotický běh ");

}

Při volání metody behej na objektu třídy Robot se nyní použije překrytá metoda vypisující řetězec „robotický běh“. Není ovšem možné zaměňovat překrývání metod s jejich přetěžováním. **Při přetížení metody jsou shodné názvy, ale musí se lišit parametry. Při překrytí metody jsou deklarace naprosto shodné a liší se prováděné příkazy.**

Zdroje:

ROUBALOVÁ, Eliška. *Java bez předchozích znalostí*. Brno: Computer Press, 2015. ISBN 978-80-251-4572-2.

BAYER Tomáš. *Objektově orientované programování* [online]. [cit. 2019-03-19]. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Prog2/prog2\_0.pdf

ČÁPKA David. *Úvod do objektově orientovaného programování* [online]. [cit. 2019-03-19]. Dostupné z: https://www.itnetwork.cz/java/oop/java-tutorial-uvod-do-objektove-orientovaneho-programovani

In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-19]. Dostupné z: https://cs.wikipedia.org/wiki/Java\_(programovací\_jazyk)