Prednáška 6:
Dedičnosť. Hierarchia tried

Ján Lang

kanc. 4.34, lang@fiit.stuba.sk, http://www2.fiit.stuba.sk/~lang/zoop/

Ústav informatiky, informačných systémov a softvérového inžinierstva

Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave
24. október 2023

Znovu použitie tried

- Princíp znovu použitia programového kódu
- Použitie existujúceho kódu - vytvorením novej triedy, aplikáciou existujúcej triedy, ktorú už niekto naprogramoval
- Spôsob znovu použitia bez zásahu do existujúceho kódu (rešpektovanie princípu zapuzdrenia)
- Zatiaľ dva možné spôsoby:
  - Vytvorenie inštancie existujúcej triedy v novej triede – **skladanie**. Nová trieda je poskladaná z inštancií existujúcej triedy/tried. Jednoduché znovu použitie funkcionality, nie formy
  - Vytvorenie novej triedy ako typu už existujúcej triedy. Doslovné prevzatie formy existujúcej triedy a jej ďalšie možné rozšírenie bez modifikácie existujúcej triedy - dedenie
- ...príklad na prednáške: Bank account

Znovu použitie tried

- Princíp znovu použitia programového kódu
- Použitie existujúceho kódu - vytvorením novej triedy, aplikáciou existujúcej triedy, ktorú už niekto naprogramoval
- Spôsob znovu použitia bez zásahu do existujúceho kódu (rešpektovanie princípu zapuzdrenia)
- Zatiaľ dva možné spôsoby:
  - Vytvorenie inštancie existujúcej triedy v novej triede – skladanie. Nová trieda je poskladaná z inštancií existujúcej triedy/tried. Jednoduché znovu použitie funkcionality, nie formy
  - Vytvorenie novej triedy ako typu už existujúcej triedy. Doslovné prevzatie formy existujúcej triedy a jej ďalšie možné rozšírenie bez modifikácie existujúcej triedy - **dedenie**
- ...príklad na prednáške: A, B, C, D?, porovnanie možností skladania vs. dedenia

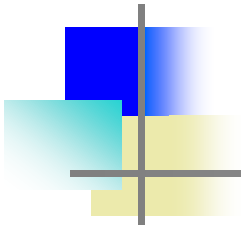
Skladanie vs. Dedenie

- Inheritance should only be used, if the subclass can be used behaviorally equal to the Superclass (Liskov substitution principle - *Kde možno použiť inštanciu predka, tam musí byť možné použiť inštanciu potomka*)

```
class DB_Connect { }

Class MySQL_DB_Connect extends DB_Connect { }

Class MySQL_DB_Connect {
 DB_Connect db;
}
```



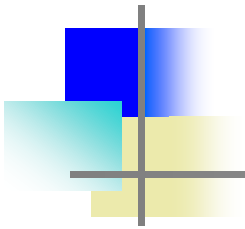
Skladanie vs. Dedenie

- "A Car is a Wheel" (subclassing) doesn't make sense,
- "A Car has a Wheel" (instantiation) works ☺

Dedenie

```
1  package sk.stuba.fiit.mesto;
2
3  public class Obyvatel {
4
5  }
6
```

- To, čo skladanie nedovoľuje, je zmena spektra/štruktúry vlastností triedy. Prispôsobenie formy.
- Vytvorenie novej triedy ako typu už existujúcej triedy. Doslovné prevzatie formy existujúcej triedy a jej ďalšie možné rozšírenie bez **priamej** modifikácie existujúcej triedy – **dedenie**
- Dedičnosť predstavuje možnosť pridať k základnej (rodičovskej) triede ďalšie vlastnosti a vytvoriť tak odvodenú triedu (potomka)
- Vzniká vzťah Nadtyp-Podtyp. Spôsob vytvárania hierarchie
- Používa sa v prípadoch, keď od všeobecnej triedy chceme odvodiť konkrétnejšiu
- Podtyp môže získanú štruktúru Nadtypu **rozšíriť** o nové **atribúty** a **metódy** prípadne **zmeniť implementáciu** zdedených metód

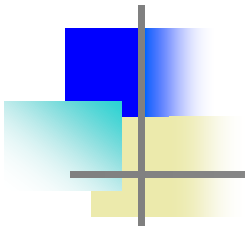


Dedenie

- Odvodenie povahy od svojho Nadtypu
- Dedenie nám v odvodenej triede umožňuje:
  - ponechať a využívať všetko z rodičovskej triedy
  - doplniť, čo v rodičovskej triede nebolo a v odvodenej chýba
  - zmeniť to, čo z rodičovskej triedy nevyhovuje
- Kľúčové slovo `extends` nasleduje názov triedy Nadtypu

```
Class Obyvatel extends Clovek {

}
```

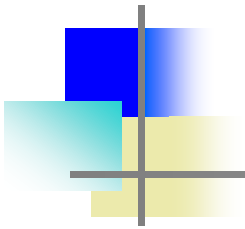


Príklad

- `Obyvateľ` - špecifická trieda typu `Clovek`
- `Clovek, Datum, Poloha, Domcek` - nové triedy v našom projekte

- Použitie už vytvoreného kódu – triedy `Datum, Poloha, Domcek` skladaním - t.j. - vytvorením objektov už existujúcej triedy (`Datum, Poloha, Domcek`) v rámci novej triedy (`Clovek`)

- Použitie už vytvoreného kódu – triedy `Clovek` dedením - t.j. - vytvorením objektov už existujúcej triedy (`Clovek`) dedením vlastností v triede (`Obyvateľ`)
- Rozšírenie typu `Clovek` typom `Obyvateľ`
- `Obyvateľ` je podtypom, `Clovek` nadtypom



Príklad

- Parametrický konštruktor v triede `Clovek`
- Riešenie napr. bezparametrický konštruktor v triede `Clovek`
- V uvedenom príklade trieda `Obyvatel` dedí od triedy `Clovek`, čo je uvedené pomocou kľúčového slova `extends` za názvom triedy `Obyvatel`
- Trieda `Obyvatel` zdedila od triedy `Clovek` premenné `meno`, `priezvisko` a `datumNarodenia`.

Dedenie

- Explicitným uvedením konštruktora zaniká implicitný konštruktor (bezparametrický konštruktor)
- ...príklad priamo na prednáške (A, B, C, D triedy)

```
A() {
   System.out.println("bol volaný explicitný
konštruktor A");
}
```

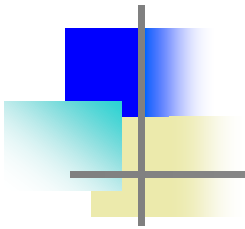
- Pri volaní konštruktora potomka sa vždy musí volať konštruktor rodiča. Ak neexistuje implicitný konštruktor rodiča je nutné explicitne volanie konštruktora rodiča v tele konštruktora potomka

Príklad

```
public class B extends A {

int premenna;

B(int i) { // toto ruší implicitný konštruktor
    this.premenna = i;
    System.out.println("bol volany explicitny
      konstruktor B");
    }
}
```

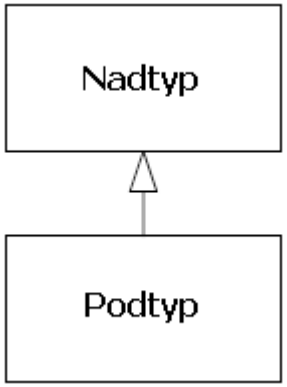


Dedenie

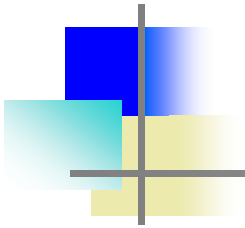
- Pri volaní konštruktora potomka sa vždy musí volať konštruktor rodiča
- Môžu však nastať dva prípady:
  - Keď je v rodičovskej triede implicitný konštruktor alebo je medzi konštruktormi v rodičovskej triede konštruktor bez parametrov, konštruktor v triede potomka môže byť implicitný. A keď konštruktor v triede potomka nie je implicitný, nemusíme v ňom volať žiadny konštruktor z rodičovskej triedy, pretože prekladač si sám doplní implicitný konštruktor, či konštruktor bez parametrov
  - Keď majú v rodičovskej triede všetky konštruktory aspoň po jednom parametre, konštruktor v triede potomka musí existovať a ako svoj prvý príkaz musí volať pomocou kľúčového slova super niektorý konštruktor z rodičovskej triedy.

Dedenie

- Vzťah: rodič – potomok, nadtyp – podtyp, podtrieda – nadtrieda, bázová trieda – odvodená tireda, base class – derived class, subclass – superclass,...

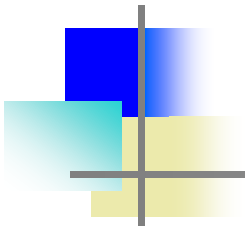


- Šípka v UML diagrame smeruje z Podtypu do Nadtypu
- Príklad UML class diagramu (shape, circle, square, triangle) v TiJ, *Chapter 1: Introduction to Objects*



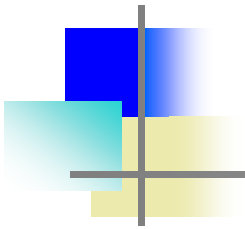
Dedenie

- Modifikátory prístupu
  - private – atribúty Nadtypu, ktoré majú ostať nedostupné v Podtype
  - protected - atribúty Nadtypu, ktoré majú ostať dostupné v Podtype a v celej hierarchii dedenia, tiež v balíku



Dedenie

- Typ jasne definuje jeho rozhranie
- Typ poznáme podľa správ, ktoré mu môžeme poslať
- Keďže Podtyp disponuje tým istým spektrom metód ako jeho Nadtyp. Môžme povedať, že Podtyp je v podstate ten istý typ ako jeho Nadtyp
- Trieda Podtypu bez zásahu do zdedeného správania má identické správanie ako jej Nadtyp - čo je **nezaujímavé!**
- Riešenie – diferenciácia novej triedy, Podtypu od svojho Nadtypu
  - Pridaním nových metód do Podtypu (Nadtyp však týmto nebude disponovať...). Láka k tomu aj samotný význam kľúčového slova `extends` - rozširuje
  - Zmenou správania sa – zmenou implementácie metód Nadtypu. Tejto diferenciácii hovoríme **prekonanie** (overriding) metód

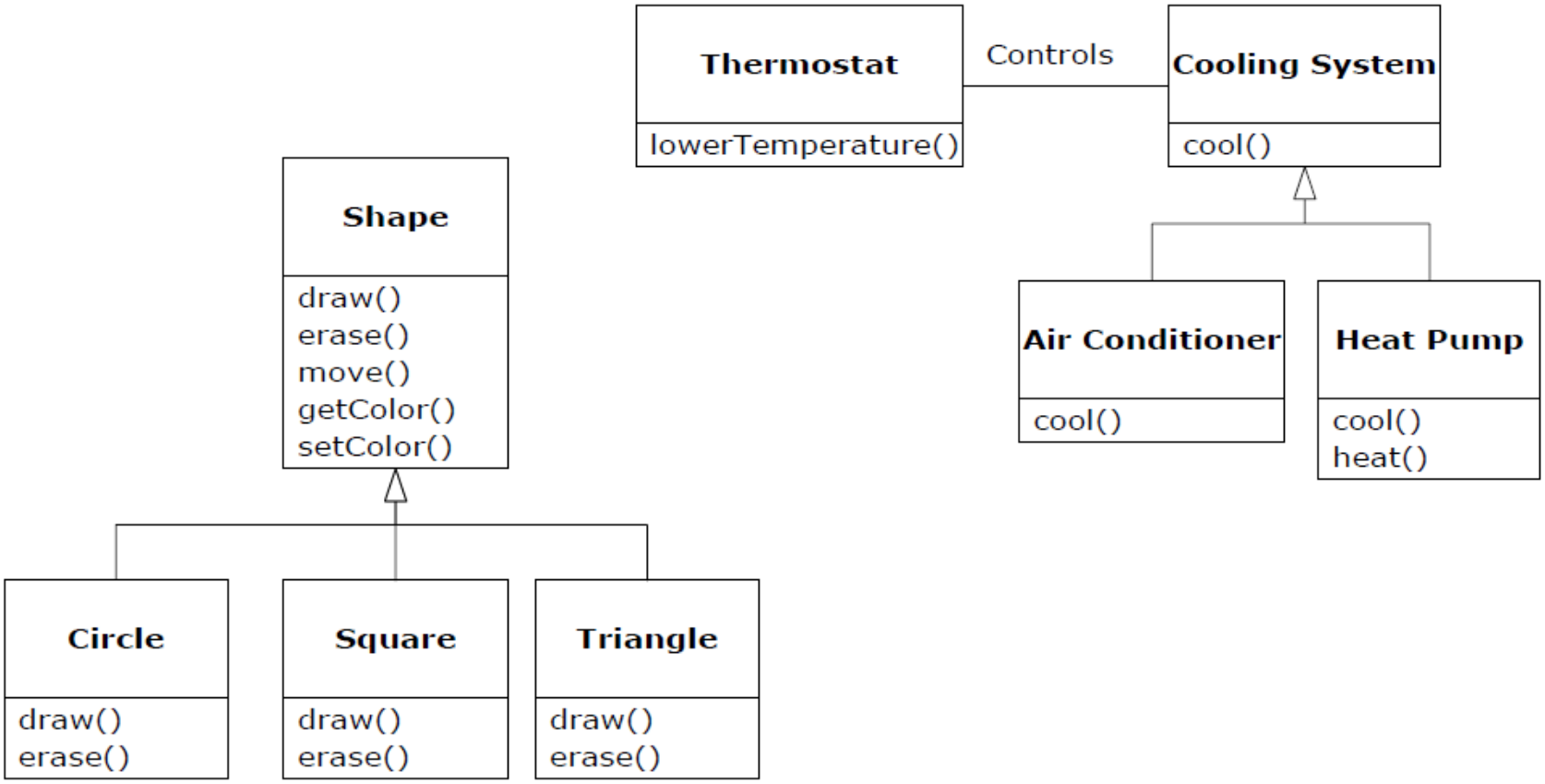


Dedenie

- Typ jasne definuje jeho rozhranie
- Typ poznáme podľa správ, ktoré mu môžeme poslať
- Keďže Podtyp disponuje tým istým spektrom metód ako jeho Nadtyp. Môžme povedať, že Podtyp je v podstate ten istý typ ako jeho Nadtyp
- Trieda Podtypu bez zásahu do zdedeného správania má identické správanie ako jej Nadtyp - čo je **nezaujímavé!/nepotrebné!**
- Riešenie – diferenciácia novej triedy, Podtypu od svojho Nadtypu
  - Pridaním nových metód do Podtypu (Nadtyp však týmto nebude disponovať...). Láka k tomu aj samotný význam kľúčového slova `extends` - rozširuje
  - Zmenou správania sa – zmenou implementácie metód Nadtypu. Tejto diferenciácii hovoríme **prekonanie** (overriding) metód

Dedenie

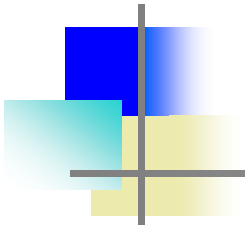
- UML reprezentácia diagramu tried



Skladanie vs. Dedenie

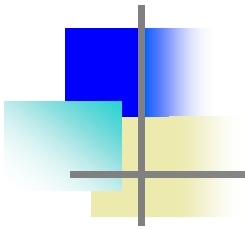
- Keď sa pri novej triede rozhodujeme, či použiť skladanie alebo dedičnosť pomocou už existujúcej triedy, môžeme vyskúšať test pomocou slovíčok **JE** alebo **MÁ**
- Ak nová trieda MÁ existujúcu triedu, používame skladanie, ako v príklade: `Clovek` **MÁ** `DatumNarodenia`
- Ak nová trieda JE existujúcou triedou, používame dedičnosť, ako v príklade: `Clovek` **JE** `Obyvatel`
- ...uveďte vlastnépríklady dokumentujúce túto skutočnosť. Možno z vlastného Zadania

- Skladanie a dedičnosť sa často kombinujú, teda v zložitejších triedach sa vyskytujú spoločne



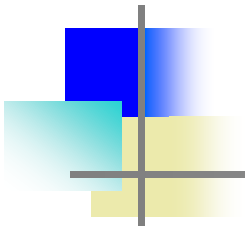
Preťaženie metód pri dedení

- V prípade, že sa rovnomenné metódy (zdedené a pridané) v Podtype líšia v parametroch dochádza k preťaženiu
- ...príklad na prednáške.



Prekonávanie (overriding) metód

- V prípade, že sa rovnomenné metódy (zdedené a novo pridané) v Podtype nelíšia v parametroch dochádza k prekonaniu
- Deklarácia nestatickej metódy rovnakej signatúry v Podtype prekonáva (overrides) pôvodnú metódu Nadtypu
- ...príklad na prednáške.

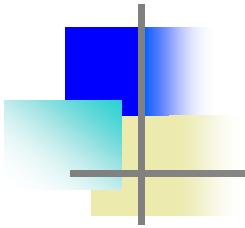


Kľúčové slovo `super`

- Pomocou kľúčového slova môžeme v budovanej hierarchii pristupovať k skrytým atribútom a prekonaným metódam
- ...príklad na prednáške.

Kľúčové slovo `final`

- Deklaráciu konštantných atribútov – po inicializácii nemožno zmeniť ani v Podtype. Nemožnosť skryť atribút vytvorením rovnomenného v Podtype
- `final` metódy - môže sa stať, že považujeme niektorú metódu triedy za tak dôležitú, že nechceme, aby ju bolo možné v zdedených triedach skryť. Tomu môžeme zabrániť tým, že pred názov metódy vložíme kľúčové slovo `final`
- `final` triedy - taká trieda sa potom nedá zdediť, teda nemôže byť rodičovskou triedou. A samozrejme, že keď je trieda `final`, tak jej metódy sa nedajú skryť, keďže trieda sa nedá zdediť

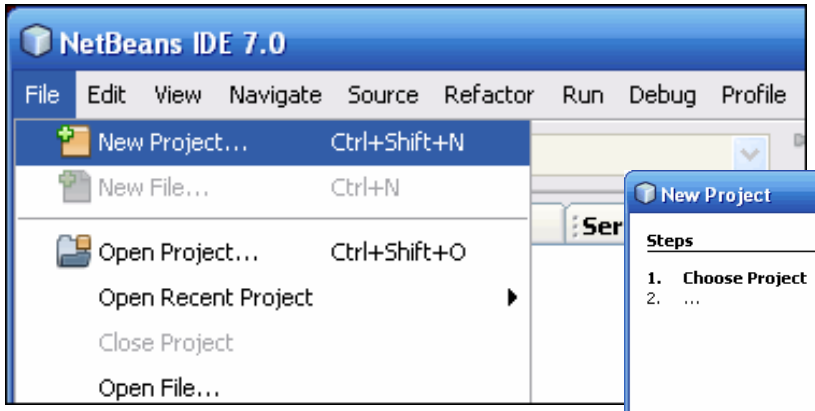


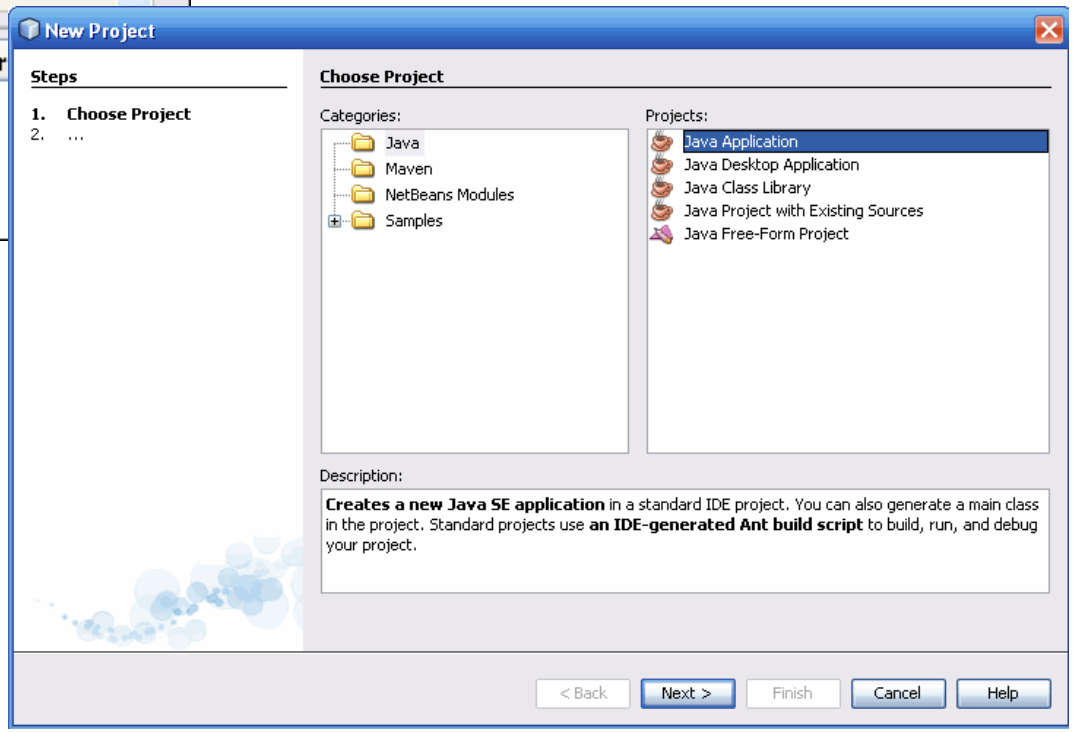
Príklad

- …na prednáške:
  - Preťaženie pri dedení
  - Skrývanie atribútov tried
  - Modifikátor prístupu protected (prvok je dostupný každému Podtypu v celej hierarchii dedenia), prvok je dostupný triedam v tom istom balíku
  - Vzťah Is-a vs. is-like-a (*TIJ Chapter 1: Introduction to Objects)*

NetBeans IDE

- https://netbeans.org/downloads/
- V nástroji NetBeans vyberieme File | New Project

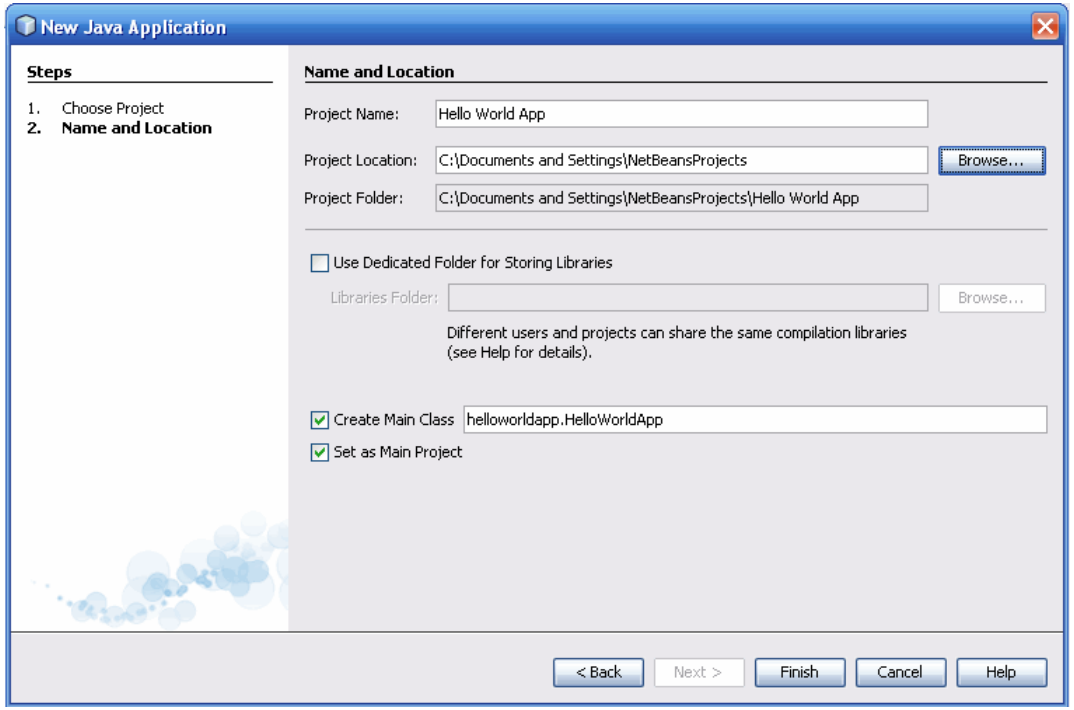




- vyberieme Java Application

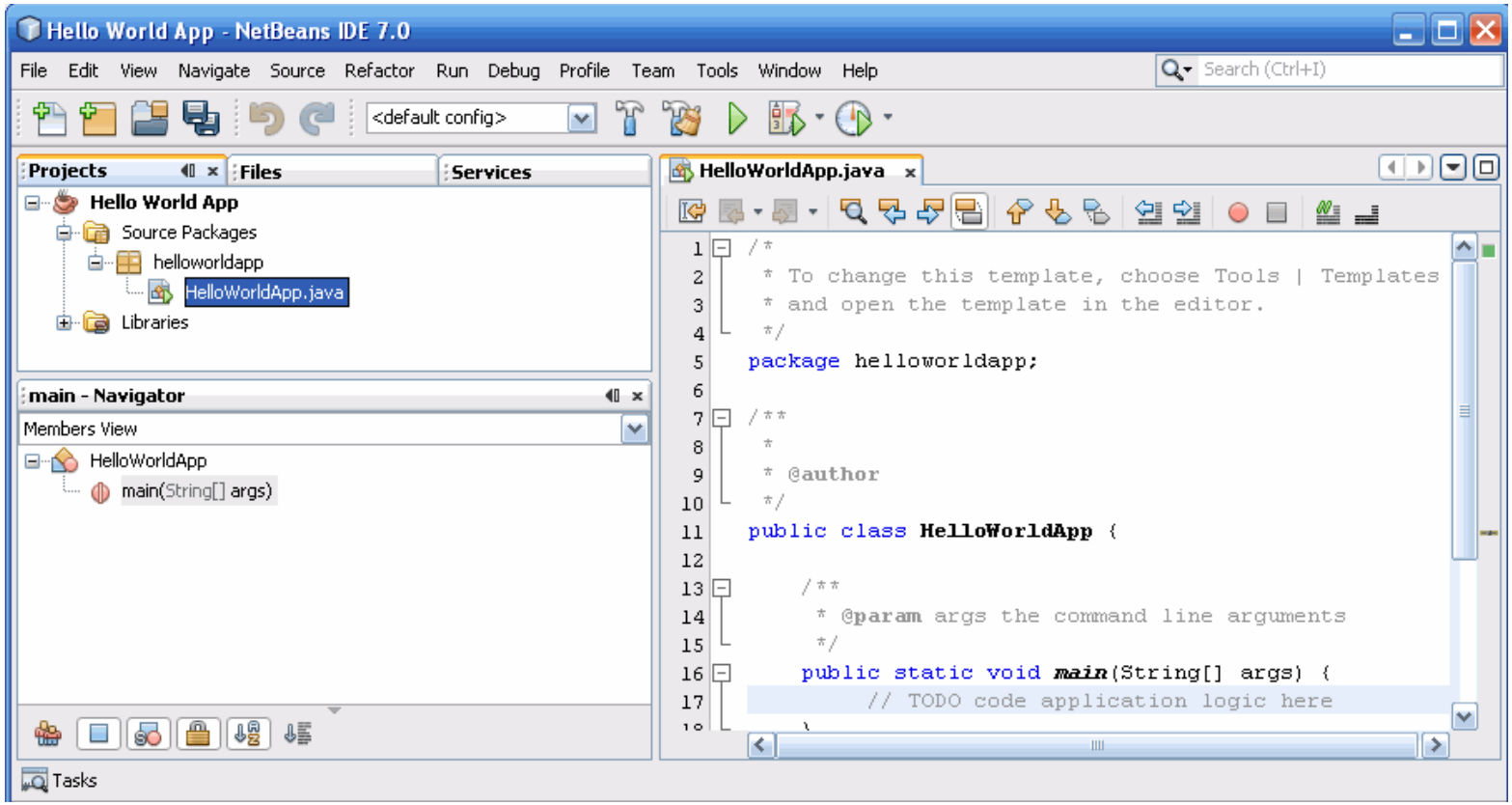
NetBeans IDE

- V sekcii „Name and Location“ vyplníme nasledujúce položky:
  - V „Project Name“ vypíšeme: napr. `Hello World App.`
  - V „Create Main class“ vypíšeme: `helloworldapp.HelloWorldApp.`
  - Môžeme nechať zaškrtnuté políčko „Set as Main Project“.



NetBeans IDE

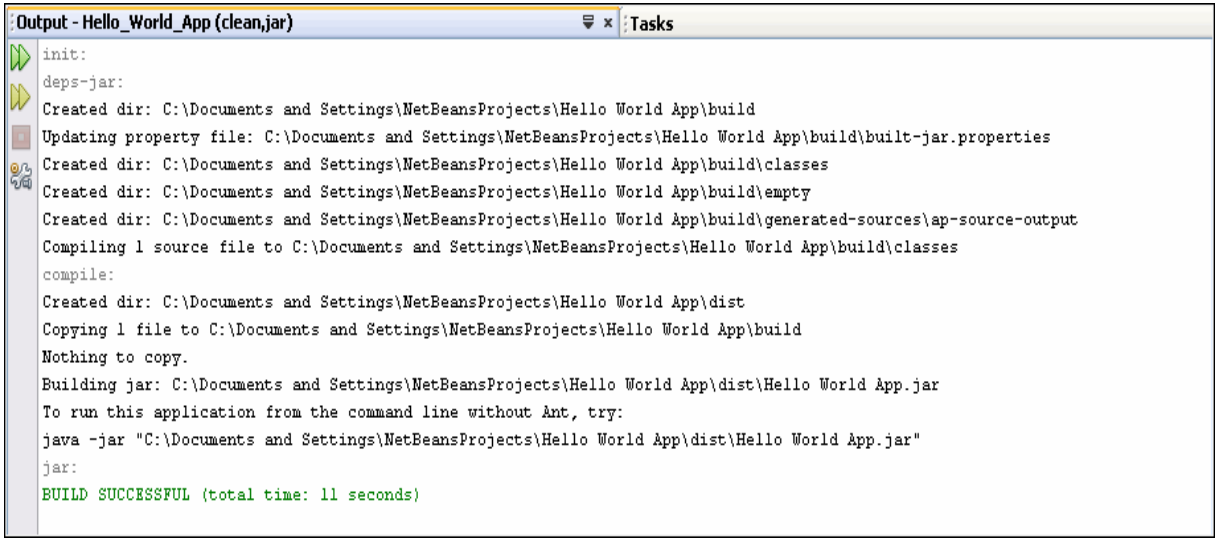
- Otvorí sa nám okno prostredia



NetBeans IDE

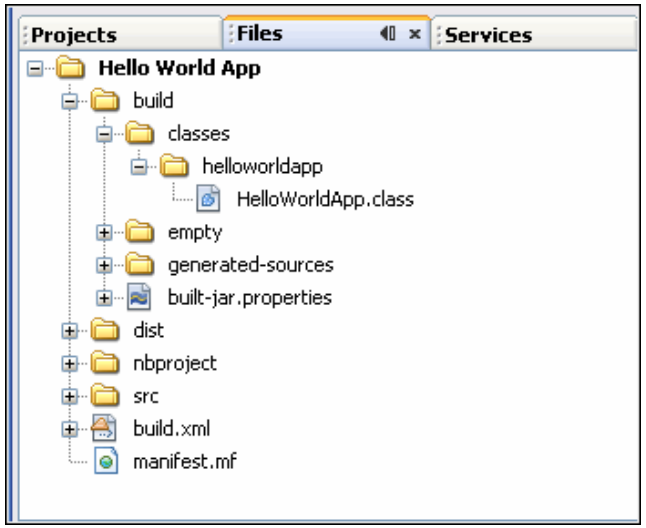
Kompilácia zdrojového súboru do .class

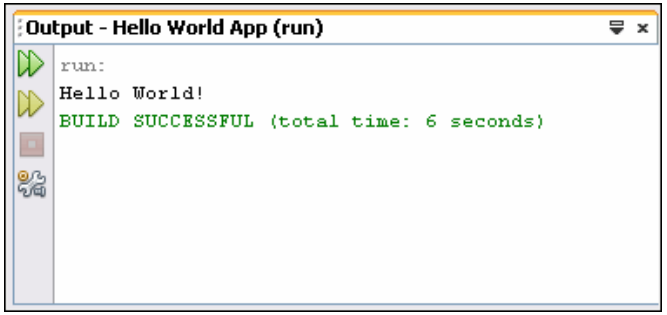
- Zdrojový kód skompilujeme cez Run | Build Main Project alebo Build Project (MenoProjektu). (kláves **F11**)
- Môžeme si priebeh kompilácie pozrieť v Output okne, ktoré vyvoláme cez (CTRL + 4) alebo Window | Outpu | Output.



NetBeans IDE

- Keď sme úspešne skompilovali súbor, bol vytvorený bytecode s príponou class (HelloWorldApp.class).
- Môžeme si ho pozrieť na karte file v navigácii





- Spustenie programu cez Run | Run project (Meno projektu). (kláves F6)

TODO

- Admin schválil Vašu žiadosť o rezerváciu miestnosti -1.58 (U120) (BA-FIIT-FIIT) v termíne 07.11.2023 16.00 - 17.50 hod. - test

- Aj vy môžete pomôcť vylepšiť tento predmet študentom pre nasledujúci akademický rok. Vaše odporúčanie, komentár či otázka.