Tvorba SW

- Výroba/tvorba/vývoj/implementácia – do veľkej miery synonymá
- Tvorí sa nové SW riešenie? Do akej miery je SW riešenie vyskladané z existujúcich riešení?
- Produkt verzus projekt
  - Produktovo orientovaná tvorba
    - Bez priamej spolupráce s budúcimi užívateľmi, spravidla hromadný predaj
  - Projektovo orientovaná tvorba
    - Vývoj na zákazku , prispôsobenie produktu, ...
    - Slovenský problém – Štátne zakázky – spravidla predražené riešenia „šité na mieru“
- Tvorba SW sa riadi autorským zákonom (Zákon č. 185/2015 Z. z.)
  - Autorovi počítačového programu prináleží k tomuto programu **autorské právo**. Autorské právo zahŕňa výhradné osobnostné práva a výhradné majetkové práva k počítačovému programu.
  - **Zamestnanecké dielo** - Majetkové práva autora k zamestnaneckému dielu vykonáva vo svojom mene a na svoj účet zamestnávateľ, ak nie je dohodnuté inak. Pri výkone majetkových práv autora k zamestnaneckému dielu zamestnávateľom nesmie autor udeliť tretej osobe súhlas na použitie tohto diela a autor je povinný sám sa zdržať výkonu majetkových práv k tomuto dielu.
  - vývojár(i) má (majú) stále autorstvo (autorské práva)
- Vlastník SW (držiteľ majetkových práv) SW môže poskytovať na základe zmluvy (licencie)
  - ak používate časti SW druhých, pozor na licenčné podmienky! (GPL, ...)
  - Pozor, čo sa týka aj „voľne šíriteľného SW“ ak pracujete na komerčnom projekte
  - Pozor, čo sa týka licencovaného SW a vy ho chcete použiť na projekte „voľne šíriteľného SW“

Prečo sa učiť o tvorbe SW ?

- Z pohľadu zadávateľa, aby bol schopný
  - sformulovať zadanie
  - prebrať riešenie
  - uviesť riešenie do chodu
  - prevádzkovať,
  - ...
- Z pohľadu realizátora
  - aby bol schopný zrealizovať zadanie za sľúbených cenových, časových, funkčných, ... podmienok
  - aby nesľúbil čo nevie dodržať ...
  - resp. aby mu vyšiel obchodný plán, aby dobre nastavil obchodné podmienky, očakávania, marketing atď ...
- ...

Obe strany majú rôzne roly, ktoré do procesu tvorby vstupujú ...

- obchodník
- projektový manažér
- analytik
- architekt
- vývojár
- tester
- pracovník prevádzky
- pracovník podpory
- ...

Keď to celé sami nezažijete, váš odhad a súdnosť je prudko zhoršená

(na tomto predmete nie je cieľom natlačiť do Vás skúsenosti ale ukázať cestu a pomôcť kalibrovať odhad.)

Čo je potrebné zohľadniť pred/pri/po tvorbe SW ?

Sumarizácia oblastí (platné pre obe strany)

- Legislatíva
  - autorstvo, SW licencie, patenty, ... ak niečo tvorite/poskytujete musíte
    - zohľadniť vstupy (treťostranný SW, podmienky použitia, ...)
    - zohľadniť proces tvorby
    - definovať práva na používanie
  - legislatíva v oblasti telekomunikácií (http://www.zakonypreludi.sk/obor/4236293)
  - legislatíva v oblasti informačných systémov (http://www.zakonypreludi.sk/obor/4235887)
    Informatizácia verejnej správy http://www.informatizacia.sk/standardy-is-vs/596s
  - ...
- Technologické závislosti (Technológie, integrácia, ...)
- Projektové závislosti (Procesy, riadenie projektu, súčinnosti...)
- Prevádzka (prevádzkový model, ...)
- Obchodný model
- ...

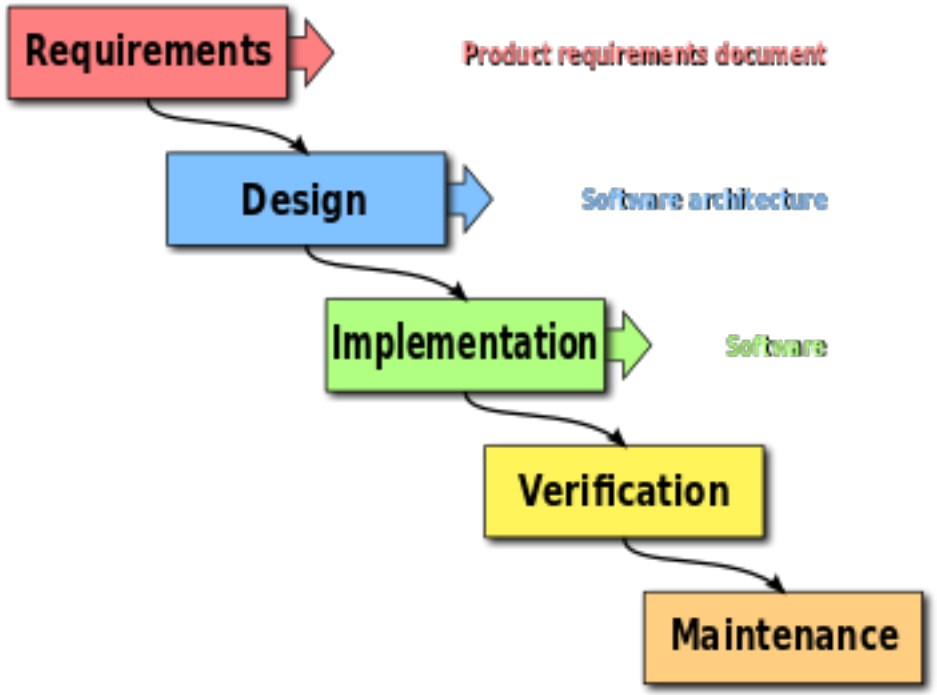
Proces vývoja

- **Vývoj SW nie je veda**, stavia na skúsenostiach a overených činnostiach (best practices)
- Proces vývoja SW – existujú viaceré metodológie ako postupovať
  - Vodopádový (základný)
  - Iteratívny/inkrementálny (UP – Unified process, RUP – Rational UP, …)
  - Prototypovanie
  - Špirálový
  - RAD (rapid application development)
  - Extrémne programovanie
  - Agilný vývoj (SCRUM, …)
- Existujú štandardy:
  - ISO12207 – životný cyklus softwéru
  - ISO15288 – životný cyklus systému
- Typické **fázy procesu vývoja** pri použití vodopádovej metodológii:
  - Zber a analýza požiadaviek
  - Návrh riešenia
  - Vývoj/implementácia
  - Testovanie
  - Integrácia
  - Nasadenie
  - Údržba
- Pri iteratívnom prístupe je process rozdelený na iterácie, ktorých výsledkom sú vydané verzie SW (releasy)

Vodopádový prístup

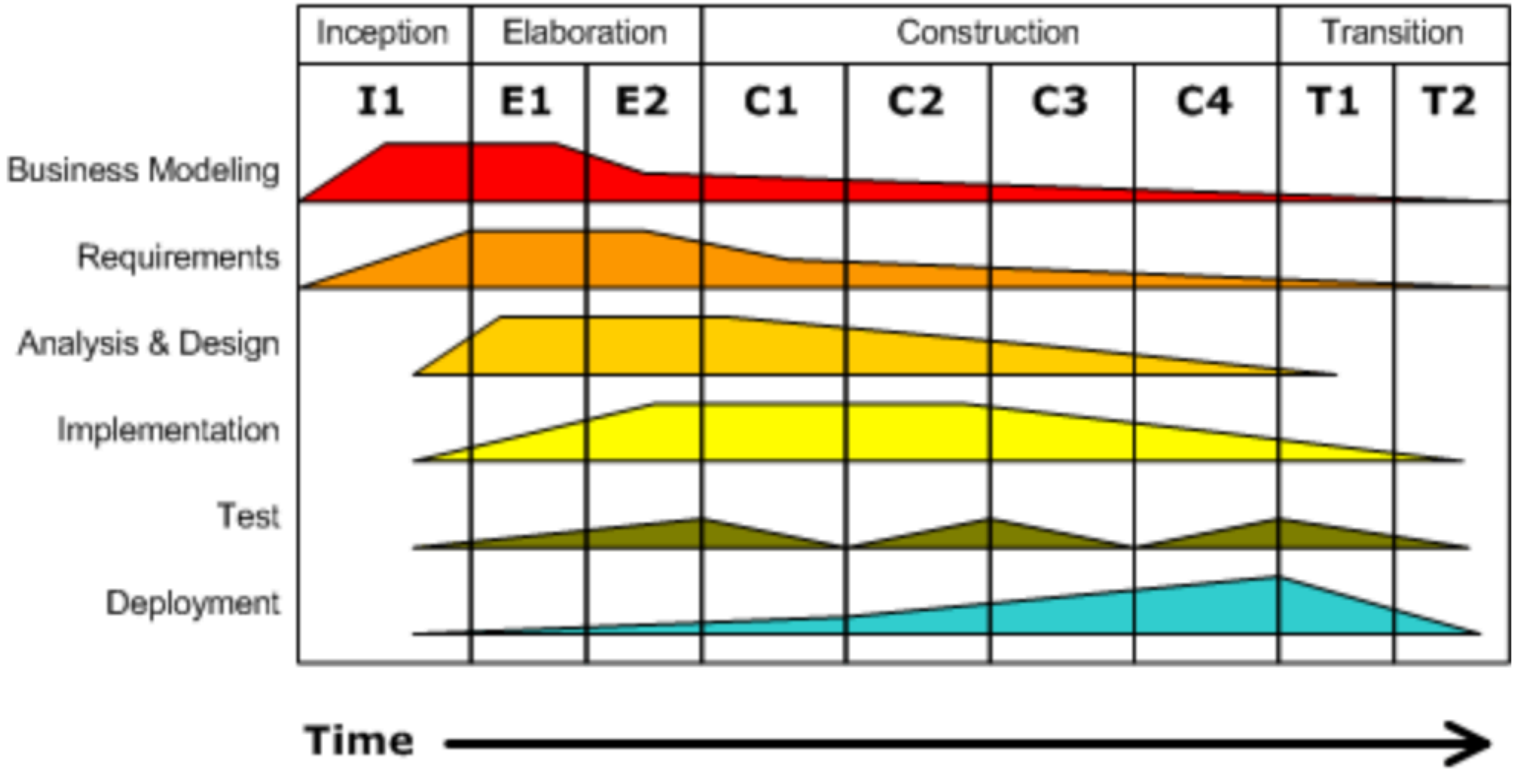
(Winston W. Royce 1970)

1. Zber požiadavôiek (requirements) – požiadavky na riešenie zachytené v dokumente (funkčné aj nefunkčné požiadavky, systémové aj softwérove, ...)
2. Analýza a návrh (modely, biznis pravidlá, architektúra) – výstupom je návrh architektúry (t.j. napr. funkčná aj technická špecifikácia riešenia, cieľový koncept riešenia, ...)
3. Implementácia – kódovanie/vývoj, integrácia
4. Testovanie/verifikácia – odstraňovanie chýb
5. Prevádzka – inštalácia, migrácia, prevádzka, podpora prevádzky a údržby, ...



Iteratívny prístup

(číslami 1, 2, 3, 4 sú označované jednotlivé iterácie)



Fázy verzus oblasti (disciplíny)

Fázy

- **Interception:** zachytenie požiadaviek, spracovanie, odhady náročnosti, zisku, ponuka
- **Elaboration:** spracovanie požiadaviek, vytvorenie základnej analýzy, stabilného návrhu (architektúry), prototypy problematických častí, plán vývoja. podrobného návrhu riešenia (architektúra, funkčné a technické špecifikácie)na prípady použitia, návrh architektúry
- **Construction:** výroba, v jednotlivých iteráciách CN demonštrovateľné verzie, prvé nasadenie u zákazníka
- **Transition:** prechod do plnej produkcie, beta testovanie, zaškoľovanie, akceptácia

Porovnanie vodopádový model s iteratívnym

Vodopádový model

- Najstarší, zaviedol systematickosť
- Ďalšia fáza môže začať až po skončení predchádzajúcej (pri vylepšenom modeli sa môžu aj prekrývať)
- Plná dokumentácia každej fázy
- Spätnou väzbou sa dajú zmeny premietať späť
- Nedá sa paralelizovať
- Malá spätná väzba

Vodopádový model	Iteratívny model
Plánovanie zdrojov na začiatku - nepresné	Krátke iterácie vedú k lepšej kontrole a plánovaniu. Roadmapa pre celý projekt, podrobný plán len pre iterácie (max. 2 mesiace)
Riziká sú väčším prekvapením	Najrizikovejšie časti sa riešia najskôr
Integrácia a testovanie až ku koncu	Priebežná integrácia a testovanie
Zmeny nie sú vítané	Počíta sa zo zmenami, prijíma ich

SW vývoj verzus programovanie

SW vývoj je súbor viacerých aktivít, ktoré musia na seba nadväzovať

- **Programovania jednotlivých častí (modulov) SW riešenia**
- Dokumentovania
- Testovania
- Opravy chýb

Pričom výsledkom je softwérový produkt.

Teraz sa zameriame na programovanie …

Programovanie = tvorba programov
Programátor = tvorca programov, vývojár
Program = predpis v nejakom *programovacom jazyku* popisujúcom čo sa má udiať.

- Programuje sa v jazyku, ktorému rozumie programátor
- **Prekladač/kompilátor** – program, ktorý preloží iný program do jazyka, ktorému rozumie počítač
- **Interpreter** – program, ktorý iný program priamo vykonáva
- Je dôležité program písať tak, aby mu nerozumel len autor a prekladač, ale najmä
    - samotný autor po dlhšom čase
    - iní programátori

Prekladanie verzus interpretovanie

To či je jazyk interpretované, alebo prekladaný, nie je to vlastnosť jazyka samotného, je to implementačné rozhodnutie.

- Výhoda prekladania - Preložený program beží zvyčajne rýchlejšie
  - Typicky jazyky: c, c++, Pascal
- Výhoda interpretovania - Zvyčajne väčšia nezávislosť na platforme, program nie je treba upravovať
  - Typicky jazyky: perl, python, PHP, JavaScript
- **Hybridný prístup** (motivácia je mať výhody oboch predchádzajúcich)
  - Prekladač prekladá do platformovo nezávislého medzijazyka
  - Medzijazyk je optimalizovaný na to aby ho bolo možné čo najrýchlejšie interpretovať
  - Špecialny interpreter (virtuálny stroj) medzijazyk interpretuje
  - Typicky jazyky: java, c#, erlang (poznáte?)

Programovanie a paradigma

Paradigma = spôsob programovania

Základné paradigmy zahŕňajú:

1) Imperatívne programovanie – používa príkazy, ktoré menia stav programu. po krokoch popisuje postup, ktorý sa má vykonávať.
   a) Procedurálne programovanie – program je tvorený procedúrami, (sub)rutinami, funkciami atď. ...
      i) Štruktúrované programovanie – zmeny stavov sú striktne zviazané s volaním procedúr
   b) Modulárne programovanie – dôležitý je koncept modulov = nezávislé jednotky s oddelenými záujmami
   c) **OOP** – kľúčový je koncept objektov a ich správania (napr. jazyky c++, java, javascript, python, ...)
2) Deklaratívne programovanie – je popísaný želaný výsledok bez špecifikovania postupu po krokoch, ktorý k výsledku má viesť *(napr. **SQL**, HTML, **XML**, ... (v princípe všetky ML(markup lanegueges)))*
3) ...

Prečo OOP?

- Zvýšením abstrakcie (oproti štruktúrovanému programovaniu) sa dá rýchlo a spoľahlivo navrhovať a vyvíjať väčšie a komplikovanejšie projekty
- Hlavný prúd súčasného programovania
- Výborné výrazové prostriedky pre modelovanie entít (aj entít reálneho sveta)
  - Namiesto typov štruktúr a štruktúr máme triedy a objekty
  - modelujeme nielen čo entita obsahuje, ale aj čo s tým obsahom dokáže robiť
  - keď s entitou interagujeme, nemusíme nutne poznať celé jej vnútro stačí, že vieme čo a za akých okolností vie poskytnúť
  - hodnota entity nie je v tom, čo obsahuje, ale čo a ako (v akej forme, ako efektívne) vie poskytnúť svojmu okoliu a akú má pri tom spotrebu zdrojov (tento pohľad je priam filozofický)
  - entity „neexistujú vo vzduchoprázdne“ (a ak aj, také nemáme dôvod modelovať)
    - skladajú sa z iných entít
    - majú vzťah k iným entitám ...
- Príklady entít (konkrétny človek, štát, prístroj, ...)
  - Príklad pre konkrétneho človeka
    - Štrukturálny popis
      - typ štrukúry „človek“ (váha, výška, IQ, ...)
      - štruktúra „Homer“ (vek=38, výška=183, IQ=55, ...)
    - Objektový popis
      - trieda človek (váha, výška, IQ, ..., chápe požiadavku „povedz_hlavné_mesto_štátu (xy)“)
      - objekt „Homer“ (váha=38, výška=183, IQ=55, ..., na požiadavku „povedz_hlavné_mesto_štátu (xy)“ odpovedá „neviem“)
      - objekt „Leonardo“ (váha=60, výška, =160, IQ=220..., na požiadavku „povedz_hlavné_mesto_štátu (xy)“ odpovedá správne)

Prečo JAVA?

- umožňuje používať viacero paradigiem (procedurálne, modulárne, OOP, ...)
- dôraz je na OOP ale nie je to čisto objektový jazyk (ako napr. smalltalk)
- je jednoduchý jazyk
- používa hybridný prístup (prekladač + interpreter)
  - Javac – prekladač do tzv „bajtkódu“ (*.java ->*.class) [príklad: pokus0/hello]
  - Java – virtuálny stroj (Java Virtual Machine (JVM)) – interpretuje bajtkód
    - Niektoré JVM využívajú aj JIT (just in time) kompiláciu namiesto interpretácie
- je to jazyk, aj platforma. Dokonca platformy. Obsahujú JVM.
  - J2SE - Štandardná edícia (obsahuje základné knižnice. Cieľ desktop a jednoduchšie serverové aplikácie)
  - J2EE – Enterprise edícia (obsahuje ďalšie knižnice pre tvorbu rozsiahlejších aplikácií). Potrebuje J2EE aplikačný server (napr. oracle dodáva glassfish)
  - J2ME - mikro edícia (zjednodušená verzia, menej knižníc, mobilné zariadenia)
  - Java Card – platforma pre smartkarty - beží napr. na SIM kartách (viete, že SIM karta je vlastne malý počítač), má kompaktnejší bajtkód

Slabiny a sily Javy (ťažko sa oddeľujú vlastnosti jazyka a implementácie v zmysle prekladaný/interpretované kód)

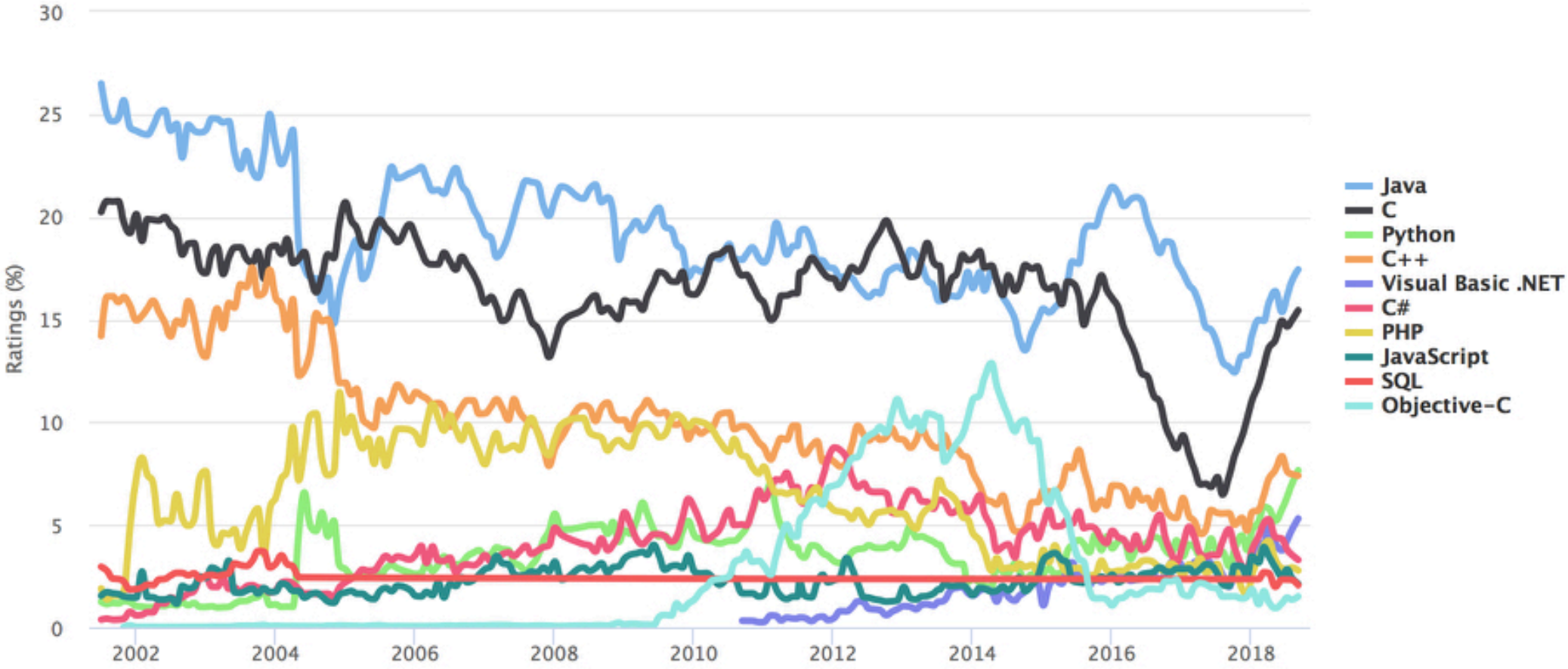
slabiny	sily
Rýchlosť v porovnaní s C, C++ Réžia objektov (z hľadiska spotreby pamati voči štruktúram v C, pričom je nemotorné sa im vyhýbať) ...	Obrovský ekosystém knižníc Multiplatformovosť ...

JAVA – informácie

- distribúcie
  - aktuálna je verzia 1.9
  - JRE – Java runtime environment – všetko potrebné pre beh programov
  - JDK – Java development kit = JRE + nástroje pre vývoj (prekladač, generátor dokumentácie, debugger, ...)
- bajtkód pre JVM nemusíme vytvorit pomocou jazyku JAVA, môžeme použiť iný (potrebujeme odpovedajúci prekladač) – napr. https://en.wikipedia.org/wiki/List_of_JVM_languages
  - python -> jython
  - javascript -> Rhino, Nashorn
  - pascal -> Oxygene, MIDletPascal
  - php –>Quercus JPHP
  - Perl -> Rakudo Perl 6
  - Visual Basic -> Jabaco
  - ...
- Popularita Javy a ostatných programovacích jazykov?
  - na ďalších stranách sú rebríčky „popularity“, hovoria ako často sú tieto jazyky používané
  - nehovoria o tom
    - ktorý jazyk je „najlepší“, ani
    - v ktorom jazyku bolo naprogramovaných najviac riadkov zdrojového kódu
  - hovorí o tom, aké zručnosti sú aktuálne požadované (na globálnom trhu)
  - hovorí o tom, že keď máte urobiť strategické rozhodnutie ohľadom voľby jazyka/platformy pre nové SW riešenie (ľudské zdroje, dostupné knižnice, dostupnosť know-how, ...)

JAVA – Popularita programovacích jazykov

[http://www.tiobe.com/tiobe-index/]



JAVA – Popularita programovacích jazykov (prehľad za 30 rokov)

[http://www.tiobe.com/tiobe-index/]

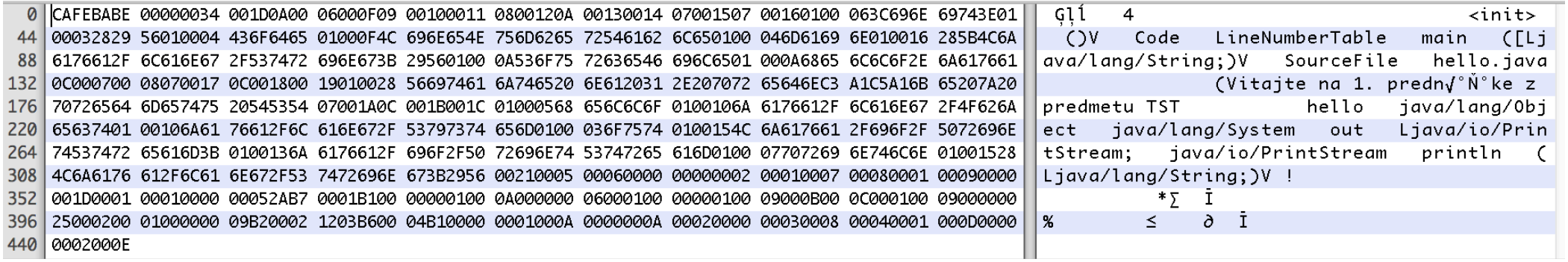
Programming Language	2018	2013	2008	2003	1998	1993	1988
Java	1	2	1	1	16	-	-
C	2	1	2	2	1	1	1
C++	3	4	3	3	2	2	4
Python	4	7	6	11	23	17	-
C#	5	5	7	8	-	-	-
Visual Basic .NET	6	11	-	-	-	-	-
JavaScript	7	9	8	7	20	-	-
PHP	8	6	4	5	-	-	-
Ruby	9	10	9	18	-	-	-
Delphi/Object Pascal	10	13	10	9	-	-	-
Perl	14	8	5	4	3	11	-
Objective-C	15	3	40	56	-	-	-
Ada	29	19	18	15	13	5	3
Fortran	30	25	22	12	5	3	15
Lisp	31	12	16	13	7	6	2

JAVA – prvý program

Textový súbor „hello.java“:

```
public class Hello {
        public static void main(String[] args) {
          System.out.println("Vitajte na 1. prednáške z predmetu TSIKT");
    }
}
```

1) Skompilovanie príkazom z príkazového riadku: „`javac hello.java`“ vytvorí súbor „Hello.class“ s bytekódom:



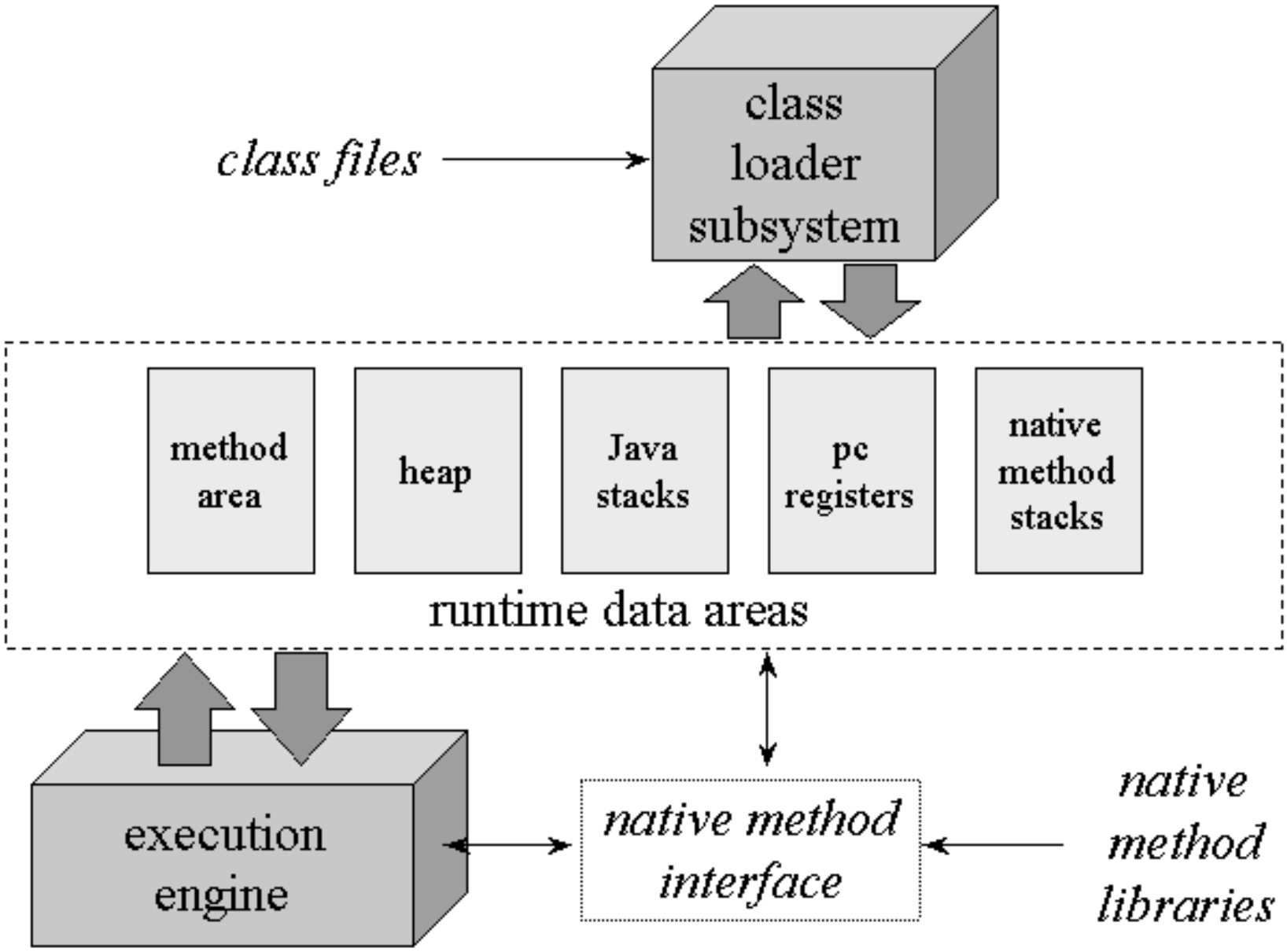
```
0   CAFEBABE 00000034 001D0A00 06000F09 00100011 0800120A 00130014 07001507 00160100 063C696E 69743E01
44  00032829 56010004 436F6465 01000F4C 696E654E 756D6265 72546162 6C650100 046D6169 6E010016 285B4C6A
88  6176612F 6C616E67 2F537472 696E673B 29560100 0A536F75 72636546 696C6501 000A6865 6C6C6F2E 6A617661
132 0C000700 08070017 0C001800 19010028 56697461 6A746520 6E612031 2E207072 65646EC3 A1C5A16B 65207A20
176 70726564 6D657475 20545354 07001A0C 001B001C 01000568 656C6C6F 0100106A 6176612F 6C616E67 2F4F626A
220 65637401 00106A61 76612F6C 616E672F 53797374 656D0100 036F7574 0100154C 6A617661 2F696F2F 5072696E
264 74537472 65616D3B 0100136A 6176612F 696F2F50 72696E74 53747265 616D0100 07707269 6E746C6E 01001528
308 4C6A6176 612F6C61 6E672F53 7472696E 673B2956 00210005 00060000 00000002 00010007 00080001 00090000
352 001D0001 00010000 00052AB7 0001B100 00000100 0A000000 06000100 00000100 09000B00 0C000100 09000000
396 25000200 01000000 09B20002 1203B600 04B10000 0001000A 0000000A 00020000 00030008 00040001 000D0000
440 0002000E
```

```
 Ģļí   4                                <init>
  ()V   Code   LineNumberTable   main   ([Lj
ava/lang/String;)V   SourceFile   hello.java
                (Vitajte na 1. predn√°Ň°ke z
predmetu TST          hello   java/lang/Obj
ect   java/lang/System   out   Ljava/io/Prin
tStream;   java/io/PrintStream   println   (
Ljava/lang/String;)V !
          *∑  İ
%       ≤    ∂  İ
```

2) Bytekód spustíme príkazom „`java Hello`“ resp. s nastavenim „classpathu“ príkazom „`java -cp . Hello`“ (tento príkaz naštartuje JVM a spusí v ňom náš bytekód)
3) Na obrazovke sa zjaví, napr:

```
radomac15:pokus0 rado$ java -cp ./ Hello
Vitajte na 1. prednáške z predmetu TSIKT
radomac15:pokus0 rado$
```

JAVA JVM - architektúra



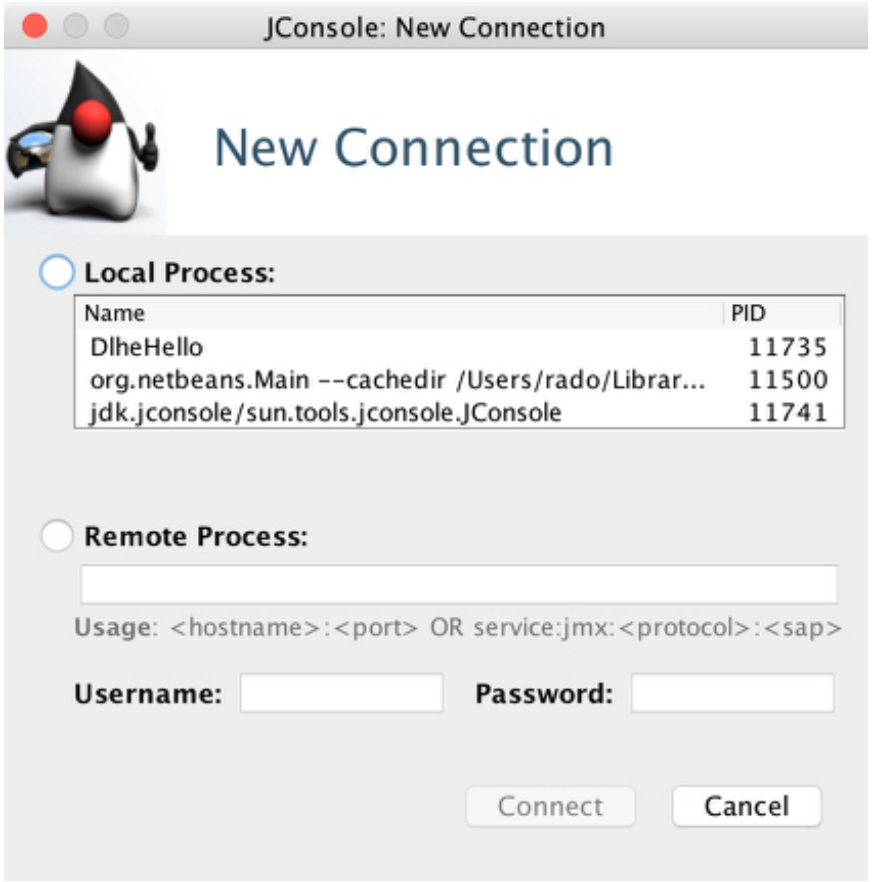
JAVA – druhý program

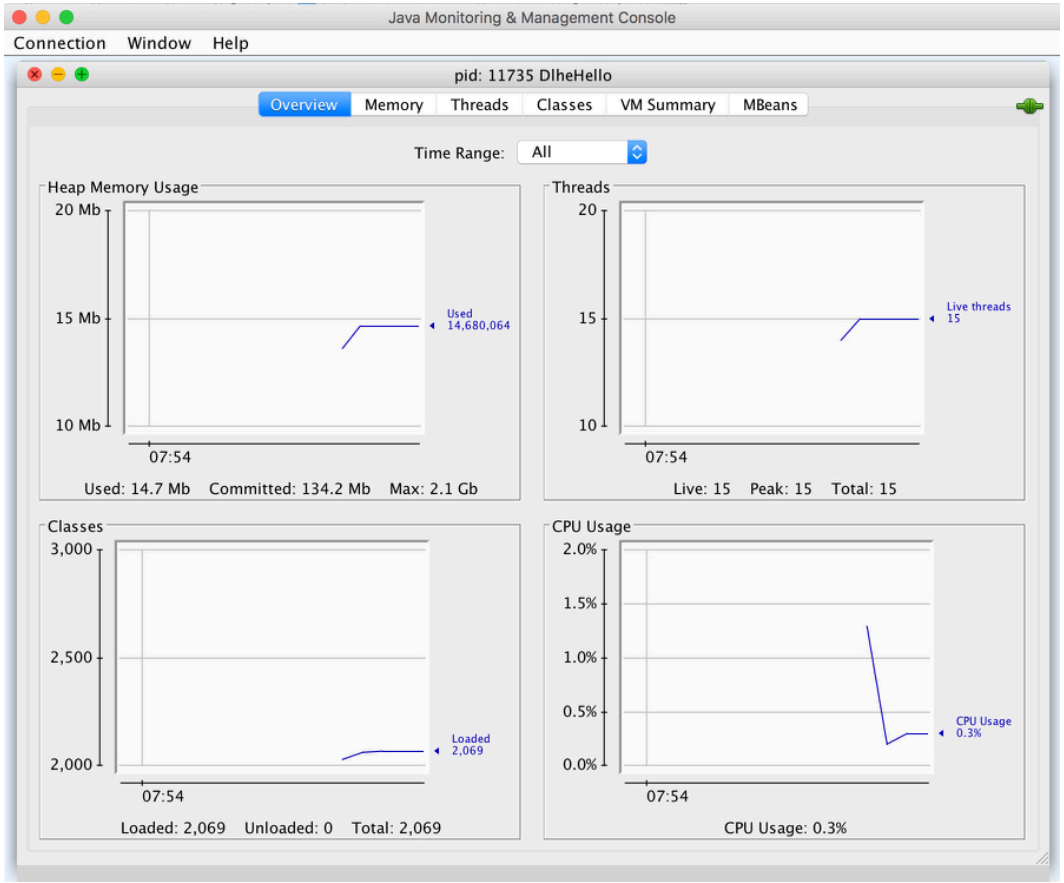
```java
public class DlheHello {
    public static void main(String[] args) {
        System.out.println("Vitajte na 1. prednáške z predmetu TSIKT");
        for(int i=100;i>0;i--) {
            System.out.println("Ešte budem bežať toľkoto sekúnd: " + i);
            try {
                Thread.sleep(1000);
            }
            catch(InterruptedException ex) {
                System.err.println("Prerušené ...");
            }
        }
        System.out.println("Koniec.");
    }
}
```

```
radomac15:pokus0 rado$ java -cp ./ DlheHello
Vitajte na 1. prednáške z predmetu TSIKT
Ešte budem bežať toľkoto sekúnd: 100
Ešte budem bežať toľkoto sekúnd: 99
Ešte budem bežať toľkoto sekúnd: 98
Ešte budem bežať toľkoto sekúnd: 97
Ešte budem bežať toľkoto sekúnd: 96
Ešte budem bežať toľkoto sekúnd: 95
^Cradomac15:pokus0 rado$
```

JAVA JVM – monitorovanie

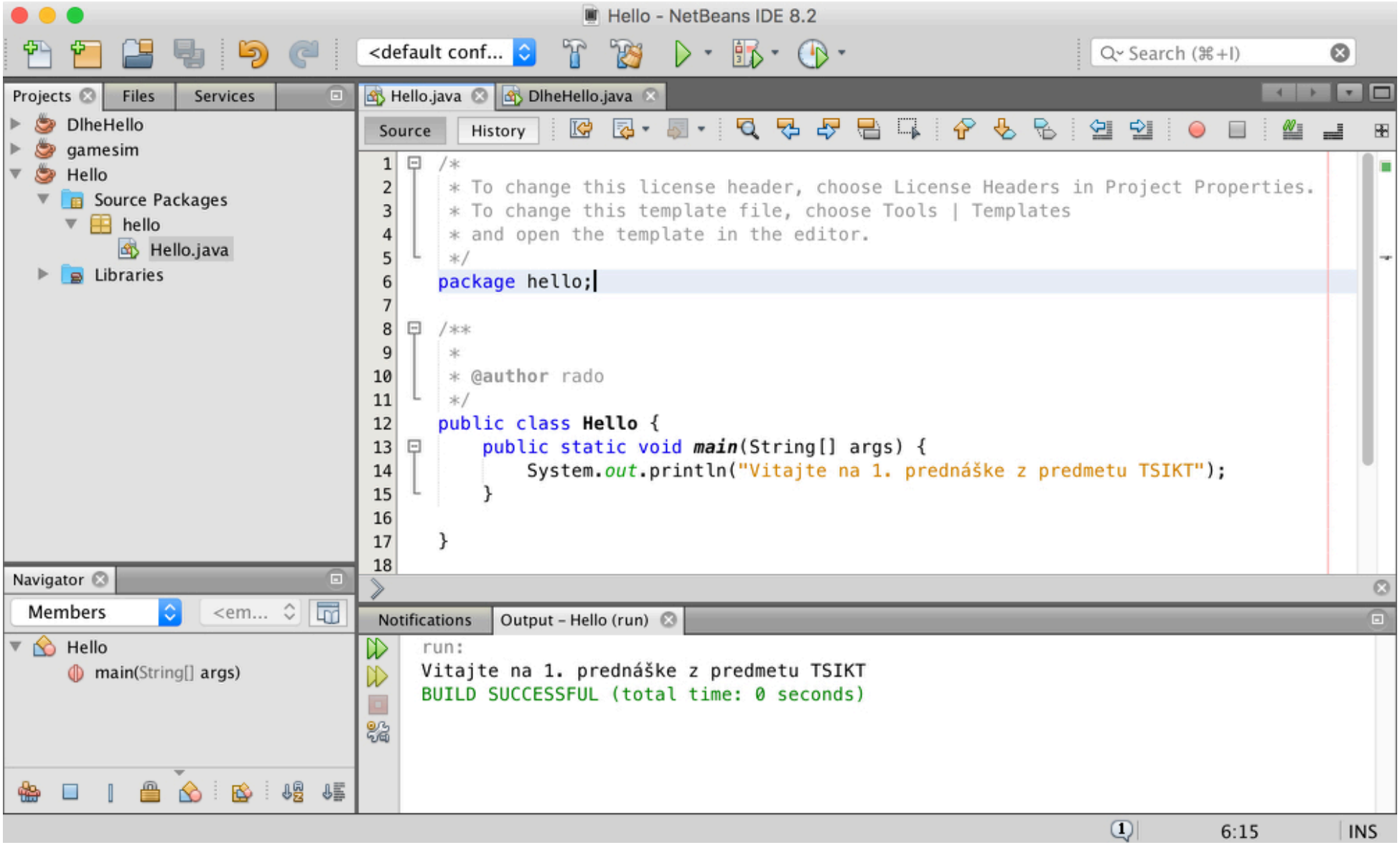
- program jconsole
- [príklad: pokus0/DlheHello]





Vývojové prostredie

- Budeme používať textový editor + príkazový riadok (už sme použili)
- IDE – Integrated development environment (budeme používať Netbeans (multiplatformový projekt napísaný v Jave s otvoreným zdrojovým kódom))



Základné prostriedky jazyka java

- zatiaľ sa cielene budeme vyhýbať triedam a objektom)
  - Keďže Java je výrazne orientovaná na OOP, nedá sa im vyhnúť úplne
- Syntax v JAVE je založená na jazykoch C a C++
  - Komentare (Jednoriadkove **//**, Viacriadkove **/* … */,** dokumentačné **/** … */**
  - Operatory … podobné ako v C
  - …
- Príklad - Netbeans prostredie
  - hello world [pokus1]

Primitívne údajové typy

- char, byte, short, int, long, float, double, boolean, void
- okrem primitívnych typov tu máme triedy … a uz sme v OOP (napr. String trieda)

Typ	**veľkosť**	**rozsah**
byte	1 byte	-128 až 127
short	2 bytes	-32,768 až 32,767
int	4 bytes	-2,147,483,648 až 2,147,483, 647
long	8 bytes	-9,223,372,036,854,775,808 až 9,223,372,036,854,775,807
float	4 bytes	približne ±3.40282347E+38F (6-7 platných číslic podľa *IEEE 754 štandardu)*
double	8 bytes	cca ±1.79769313486231570E+308 (15 platných číslic)
char	2 byte	0 až 65,536
boolean	nedefinované	true, false

Operátory

- Operátory
  - Väčšinu ich poznáte z jazyka C
  - Fungujú väčšinou len pre primitívne typy
  - Priorita je podobná ako v C
  - Druhy
    - Priradenie
    - Matematické
    - Relačné
    - Logické
    - Bitové
    - Ternárny if else
    - …

Operátor priradenia

- priraďujeme niečo premennej
- premenná má typ (java je typovo orientovaný jazyk)
    - premenná = hodnota (ak je premmenná je primitívneho typu)
    - premenné = referencia na object (pri objektoch)
    - rozsah platnosti premennej je vymedzený množinovými zátvorkami { }

Matematické operátory

- Matematické operátory sú ako v C
- Základné operácie: +, −, ∗, /,%
    - Skrátený zápis: op= (napr. +=, *=, /=, …)
- Unárne operátory: +, −
- Inkrementácia a dekrementácia: ++ a --
    - Prefixové a posfixové: c++, verzus ++c

Relačné operátory

- Zápis ako v C: ==, !=, >, <, >= a <=
- Výsledok porovnávania je typu boolean

Logické oparátory

- Zápis ako v C: &&, ||, !
- Použiteľné len pre boolean hodnoty
- Výsledok je boolean

Porovnávanie bitov

- Zápis ako v C: &, |, ^, ~
- Skrátený zápis: op= (okrem unárnho operátora ~)
- Použiteľné pre celočíselné hodnoty

Posúvanie bitov

- Zápis ako v C: <<, >>
- K tomu ešte >>> (pri posúvaní vkladá nuly bez ohľadu na znamienko, OPATRNE!)
- Použiteľné pre celočíselné hodnoty
- Použitie na char, byte, short a int dáva int výsledok;
- long dáva long výsledok
- Skrátený zápis: op=

Ternárny If–else operátor

- Zápis ako v C
- výraz

```
return i < 10 ? i * 100 : i * 10;
```

- je ekvivalentný s:

```
if (i < 10)
   return i * 100;
else
   return i * 10;
```

Špeciálne operátory

- pretypovanie

```
int i = 200;
long a = (long)i;
long b = (long)200;
```

- literály
  - sú to hodnoty zadávané priamo v programe (doslovne uvedené)

```
char c = 0xff;
float f4 = 1e-45f;
```

Modifikátory

- static
  - statickú metódu je možné volať z metódy „main“
  - statické premenné zaberajú miesto iba raz, je možné ich používať v statických metódach
  - ...
- final
  - premennú nie je možné v priebehu vykonávania meniť
  - ...
- ...

```
Static int i = 200;
...
public static void main(String[] args) {
...
```

Riadenie vykonávania

- if-else
- return
- while
- do-while
- for
- break/continue
- switch/case

If-else (C, Java):

```
if (i < 10) {
//rob niečo

} else {
//rob niečo iné

}
```

Return (pre funkciu v C / metódu v Jave):

```
int getArea() {
   return (hodnota);
}
```

Jazyk Java – modulárne programovanie

- je to možné, ale zamotávame sa do tried a objektov ešte viac.
- vytvoríme si viac modulov
  - budeme používať modifikátory: static, public/private, final
  - moduly môžu mať svoje konštanty, premenné (atribúty)
  - moduly môžu mať svoje funkcie (metódy)
- jednoduchý program sa typicky skladá zo súborov <trieda_A>.java, <trieda_B.java>
  - jedna trieda obsahuje funkciu „main“ (toto sme už mali minule):

```
public static void main(String[] args) {
...
}
```

- Príklad - Netbeans prostredie
  - hello world [pokus2_moduly]

Rozšírené modulárne programovanie – balíky

- Balíky (packages) sú podobné pridávaniu súborov v C (include)
- Použitie sa deklaruje kľúčovým slovom import
  - Napr. deklarácia použitia len jednej triedy: “import java.util.ArrayList;”
  - Napr. deklarácia použitia celého balíka: “import java.util.*;”
  - Okrem java.lang ostatné štandardné balíky Javy treba explicitne importovať
- Štruktúra a dokumentácia všetkých štandardných balíkov sa dá prehliadať na http://docs.oracle.com/javase/7/docs/api/

Java - konvencia pomenovaní

Všeobecné pravidlá:

- pokiaľ možno nepoužívať skratky, používať celé slová
- používať CamelCase / camelCase (t.j. upper camel case/ lower camel case)

Položka	Pravidlo	Príklad
Balík	• Forma reverznej domény (ako v DNS) firmy/organizácie • iba malé písmená + čísla oddelené bodkami • nepoužívať „_“	package hello;

Trieda, Rozhranie	• skladá sa z reťazcov kapitalizovaných slov (upper camel case)
	• nepoužívať „_“
	• pri triedach
	  ○ nepoužívať slovesá, iba podstatné mená s prípadnými prívlastkami ...
	  ○ iba v jednotnom čísle (okrem kolekcií
	• názov rozhrania by mal byť prídavné meno, resp. popis vlastnosti/schopnosti
		class Hello;

Metóda Premenná	• skladá sa z reťazcov slov, prvé nekapitalizované, ostatné ano (lower camel case) • nepoužívať „_“ • názov metódy má obsahovať sloveso	main(); int i;
Konštanty	• veľké písmená + čísla oddelené znakom „_“	static final int SIZE=5; static final int START_VAL=2;

Prípadne požívajte prefixy – snažte sa však každopádne opticky lokálne oddeliť lokálne premenné a parametre. Napr. lokálnym premenným dávajte prefix „my“ (napr. myCounter,...)

Java - deklarácia a definícia premenných

- Deklarácia – určujeme typ, ale nepriradujeme konkrétnu hodnotu
- Definícia – priradujeme konkrétnu hodnotu

Typy premenných

- V Jave sú 3 typy premenných
  - Lokálne premenné
    - Sú deklarované/definované v metódach a blokoch { }, len tam sú viditeľné
    - Vznikajú na zásobníku pri vstupe do metódy/bloku zanikajú pri vystúpení z metódy/bloku
    - Lokálne premenné môžu mať iba modifikátor “final”
      - Na rozdiel od C nemôžu byť static (namiesto toho sa majú používať premenné objekty/triedy)
  - Premenné objektu (inštancie triedy)
  - Premenné triedy
- Premenné mimo vyššieuvedených java nepovoľuje (napr. na začiatku modulu)