

# Úvod

Jiří Vokřínek

Katedra počítačů

Fakulta elektrotechnická

České vysoké učení technické v Praze

Přednáška 1

**B0B36PJV – Programování v JAVA**

Informace o předmětu

Programovací jazyk Java

OOP - Třídy a objekty

Význam metody `main`

Neměnitelné objekty (Immutable objects)

# Informace o předmětu

## Cíle předmětu

### Základy algoritmizace / Procedurální programování

- **Osvojit si** pohled na výpočetní prostředky a naučit se je efektivně používat *Software engineer*
  - Formulovat problém a jeho řešení počítačovým programem
  - Získat povědomí jaké problémy lze výpočetně řešit
- **Naučit se** rozkládat problémy na podproblémy
- **Získat zkušenost** s programováním *získání vlastní zkušenosti*
  - Programování v jazyku Python / C *cvičení a domácí úkoly*
- **Osvojit si** schopnost číst, psát a porozumět malým programům
- **Získat** programovací návyky jak psát
  - srozumitelné a přehledné zdrojové kódy
  - opakovaně použitelné programy

## Cíle předmětu

### Programování v JAVA

- **Prohloubit si** pohled na výpočetní prostředky a naučit se je efektivně používat *Software engineer*
  - Formulovat problém a jeho řešení počítačovým programem
  - Získat povědomí jaké problémy lze výpočetně řešit
  - Osvojit si objektově orientované programování
- **Získat zkušenost** s programováním *získání vlastní zkušenosti*
  - Programování v jazyku Java *cvičení, domácí úkoly a semestrální práce*
- **Prohloubit si** schopnost číst, psát a porozumět malým programům
- **Osvojit si** schopnost samostatně vytvořit větší programový celek *semestrální práce*
- **Získat** programovací návyky jak psát
  - srozumitelné a přehledné zdrojové kódy;
  - opakovaně použitelné programy.

## Zdroje a literatura

- Přednášky – slidy, poznámky a především **vlastní zápisky**
- Cvičení – získání praktických dovedností řešením domácích úkolů a dalších úloh

*programovat, programovat, programovat*

---

- On-line kurzy programování

*search for programming in Java*

- Knihy Java

# Knihy – Java



Učebnice jazyka Java 5. v., *Pavel Herout* KOPP, 2010, ISBN 978-80-7232-398-2



Introduction to Java Programming, 9<sup>th</sup> Edition, *Y. Daniel Liang*, Prentice Hall, 2012

<http://www.cs.armstrong.edu/liang/intro9e>



An Introduction to Object-Oriented Programming with Java, 5<sup>th</sup> Edition, *C. Thomas Wu*, McGraw-Hill, 2009

<http://it-ebooks.info/book/1908/a>



## Další literatura



Learn Object Oriented Thinking & Programming,  
*Rudolf Pecinovský* Academic series 2013, ISBN  
978-80-904661-9-7

<http://pub.bruckner.cz/titles/oop>



Java 7 – Učebnice objektové architektury pro  
začátečníky, *Rudolf Pecinovský* Grada, 2012

[http://knihy.pecinovsky.cz/ua1\\_j7/](http://knihy.pecinovsky.cz/ua1_j7/)



Java 8– Úvod do objektové architektury pro  
mírně pokročile, *Rudolf Pecinovský* Grada, 2014

Datum vydání 17.10.2014



■ <http://vyuka.pecinovsky.cz>

objektově orientované programování



# Přednášky a cvičení

- Síťové bootování a síťové domovské adresáře
- Vývoj v Javě: Owncloud – <https://owncloud.cesnet.cz>
  - Prostředí NetBeans, IntelliJ IDEA, Eclipse a Java verze 8.
  - Sestavení projektu nástrojem **ant** a **maven** <http://maven.apache.org>
- Odevzdávání domácích úkolů – Upload system <https://cw.felk.cvut.cz/upload>
- Semestrální práce – repositář systému pro správu verzí Git <https://gitlab.fel.cvut.cz>
- Práce v týmu a přesah do dalších předmětů

Pozor na rizika!

## Domácí úkoly a další úlohy

- Samostatná práce s cílem osvojit si praktické zkušenosti
- Odevzdání domácích úkolů prostřednictvím Upload system

<https://cw.felk.cvut.cz/upload>

- Nahrání (upload) archivů s nezbytnými zdrojovými soubory
- Ověření správnosti implementace automatickými testy

*detekce plagiátů*

- Podmínkou zápočtu je úspěšné odevzdání všech domácích úkolů
- Bodová ztráta za pozdní odevzdání úkolu

*Maximální počet bodů za úkol klesá s každým týdnem pozdního odevzdání*

*průběžná práce a řešení úkolů*

- Pokud něčemu nerozumíte, ptejte se cvičících

*pokud možno hned a neodkládejte na později*

- Pokud vám přijde úkolů málo, ptejte se po dalších úlohách na procvičování

## Hodnocení předmětu

Zdroj bodů	Maximum bodů	Přípustné minimum bodů
Domácí úkoly (5×5 bodů)	25	15
Semestrální práce	35	20
Zkouška	40	20

- Pro úspěšné absolvování předmětu je nutné získat **zápočet** a vykonat **zkoušku**
- Získání **zápočtu** je podmíněno odevzdáním všech domácích úkolů a odevzdáním semestrální práce

# Klasifikace předmětu

Klasifikace	Bodové rozmezí	Hodnocení	Slovní hodnocení
A	> 90	1	výborně
B	81–90	1,5	velmi dobře
C	71–80	2	dobře
D	61–70	2,5	uspokojivě
E	51–60	3	dostatečně
F	<51	4	nedostatečně

- Minimální přípustné body:

15 (úkoly) + 20 (semestrální práce) + 20 (zkouška) = 55 bodů.

# Úvod do programovacího jazyku Java

# Jazyky a Typy

- Typování proměnných a výrazů
  - **Typované** jazyky - operace typově závislé.
  - **Netypané** jazyky - libovolné operace na libovolných datech (assembly).
- Stanovení typu
  - **Statické** - při překladu, C/C++, Java.
  - **Dynamické** - za běhu, Python, Lisp.
- Typování
  - Slabé - přístup k proměnné určitého typu jako k proměnné jiného typu (assembler). Tyto jazyky jsou též nazývány jako *unsafe*.
  - Silné - kombinace typů nejsou dovoleny. Jazyky jsou nazývány jako *safe*, nebo-li typově bezpečné.

*Staticky typovaný a silně typovaný jazyk představují rozdílné dělení, přesto dochází v literatuře k záměnám významu.*

# Překlad a interpretace

- Interpreter - virtuální stroj, který vykonává přímo zdrojový kód.
- Překladač - zpracovává zdrojový kód do přímo spustitelné podoby.
- Systémové jazyky - převážně překládané, C/C++.
- Skriptovací jazyky - převážně interpretované, Ruby, Lua, Python, Perl, Tcl/Tk, shelly.

# Kompilace programu

- Kompilace zdrojového kódu
  1. předzpracování,
  2. analýza (lexikální, syntaktická, sémantická),
  3. optimalizace a generování kódu.
- Kompilace programu
  1. kompilace modulů,
  2. spojování modulů
    - statické,
    - dynamické.
- Chyby - překlad, běh, nekontrolované.
- Kontroly a efektivita.
  - Kontrola přístupu k paměti.
  - Mechanismus návratových hodnot a výjimek.



## Interpretované jazyky

- Přímá interpretace je neefektivní.
- Kompilace do binárního kódu (ne nutně nativního - *bytecode*).
- Bytecode - binární přenositelný kód.
- Just in Time (JIT), Ahead of Time (AoT) - kompilace bytecodu do nativního kódu před spuštěním.
- Dynamická kompilace - kompilace za běhu, umožňuje dynamické optimalizace (Java HotSpot VM).
  
- Většina skriptovacích jazyků používá kompilaci do byte-kódu před spuštěním.

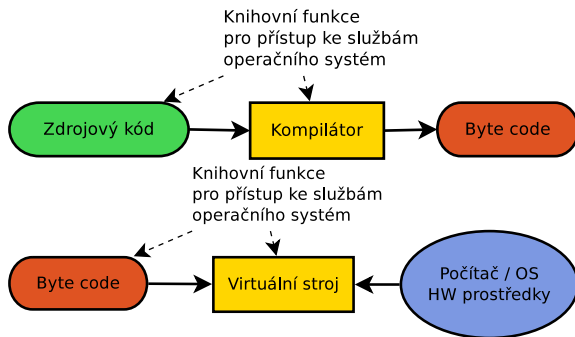
# Java

- Obecný, vyšší, imperativní (procedurální) a objektově orientovaný jazyk
- Překládaný jazyk zaměřený na přenositelnost (portabilitu) zdrojových kódů i přeložených binárních souborů
- Historie:
  - 1991 – nejdříve jako jazyk Oak
  - 1996 – Java JDK 1 (první veřejná verze)
  - 1998 – Java 2 (ver. 1.2)
  - 2004 – Java 2 (ver. 1.5) – J2SE5.0
  - 2011 – Java 7 (vydává Oracle po akvizici Sun Microsystems)
  - 2014 – Java 8
  - 2017 – Java 9 (21. září, 2017)
  - 2018 – Java 10 (březen, 2018) – Java 18.3
- Součástí základního vývojového prostředí je bohatý soubor knihovných funkcí.

*Java je relativně jednoduchý jazyk (v základní verzi) a jeho efektivní používání je spíše o znalosti knihovných funkcí.*

## Zdrojové kódy, překlad a spuštění Java programu

- Zdrojové kódy jsou zapisovány v textových souborech s koncovkou `.java`
- Zdrojové soubory jsou překládány překladačem (`javac`) do binárního kódu („*byte code*“) uložených v souborech s koncovkou `.class`
- Spuštění programu je realizováno virtuálním strojem, který poskytuje abstrakci nad operačním systémem počítače



## Java prostředí – JDK, JRE, JVM

- **JDK** (Java Development Kit) – základní vývojové prostředí, knihovny funkcí, překladač zdrojových souborů `javac`. Jeho součástí je i JRE.
- **JRE** (Java Runtime Environment) – základ prostředí Java pro spuštění programů, obsahuje virtuální stroj `java`.
- **JVM** (Java Virtual Machine) – virtuální stroj pro spuštění Java programů (`java`).
- **JAR** (Java ARchive) – archív Java souborů, typicky množiny zkompileovaných `.class` (tříd) doplněných textovým popisem (Manifest), kterou třídu spustit. Slouží pro snadnější spuštění programů o více souborech.

*V podstatě ZIP archív*

## Příklad

### Výpočet druhé odmocniny

```
1 double x = 13.0;
2 double y = 1.0;
3 int i = 1;
4
5 while(Math.abs(y*y - x) > 1e-3) {
6     System.out.println("Step " + i + " y = " + y);
7     y = (y+(x/y))/2;
8     i += 1;
9 }
10 System.out.println("sqrt(" + x + ") found in " + i + "
    steps as " + y);
```

Sqrt.java

### Kompilace a spuštění programu

```
javac Sqrt.java
java Sqrt
```

## Integrovaná vývojová prostředí (IDE)

- Nadstavba základních příkazů javac a java
- Integrují (mimo jiné) systém pro řízení překladu

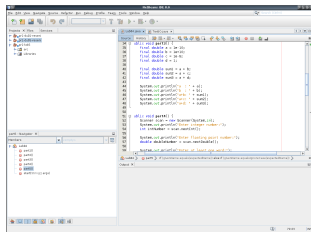
*Např. ant nebo maven*

- Zvýrazňují syntax, doplňují jména a provádějí základní kontrolu kódu
- Mezi nejznámější patří Netbeans, Eclipse a IntelliJ IDEA

<https://download.cvut.cz>

- Na cvičení je používáno prostředí Netbeans
- Import kódů do projektu
- Adresářová struktura projektu

*štábní kultura*



# Třídy, objekty a objektově orientované programování

# Charakteristika objektově orientovaného programování (OOP)

Metodický přístup řešení výpočetních problémů založený na objektovém programování.

- Abstrakce řešeného problému založena na objektovém popisu
- Objekty představují množinu dat a operací
- Objekty mezi sebou komunikují - zasílají zprávy a reagují na události
- Přístup řešení problému vychází z analogie řešení složitých problémů jak by je řešil člověk
- Základním konstruktem jsou objekty a třídy
- Vychází z objektového modelu popisu řešeného problému
- Těsnější vazba mezi analýzou a návrhem



# Objektově orientovaná analýza a návrh

- **OO analýza** se zabývá modelováním, rozbořem a specifikací problému.
  - Abstrakce reálného světa
- **OO návrh** se zabývá řešením problému.
  - Přidává softwarovou abstrakci
- Hranice mezi fází analýzy a návrhem se stírá:
  - Základní konstrukce (třídy a objekty) se používají stejné.
  - Není přesně definováno co patří do fáze analýzy a co do návrhu.
- Cílem objektově orientované analýzy a návrhu (OOAD) je:
  - popis systému reprezentovaný objektovými diagramy (statická struktura),
  - popis dynamiky a chování systému.

# Objektově orientované programování

- Základními konstrukčními prvky OOP jsou třídy a objekty  
*OOP nejsou jen třídy a objekty!*
- Umožňuje abstrakci a zobecnění řešených problémů
- Znovu použitelnost implementovaných kódů
- Kontrolu přístup k datům

OOP je přístup jak správně navrhnout strukturu programu tak, aby výsledný program splňoval funkční požadavky a byl dobře udržovatelný.

# Třídy a objekty

- **Třídy** - popisují možnou množinou objektů. Předloha pro tvorbu objektů třídy. Mají:
  - **Rozhraní** – definuje části objektů dané třídy přístupné zvenčí
  - **Tělo** – implementuje operace rozhraní
  - **Datové položky** –(statické) proměnné – společné všem instancím dané třídy
- **Objekty** - reprezentují základní entity OO systému za jeho běhu.
  - Mají konkrétní vlastnosti a vykazují chování
  - V každém okamžiku lze popsat jejich stav
  - Objekty se v průběhu běhu programu liší svým vnitřním stavem, který se během vykonávání programu mění
- *Každý objekt při svém vytvoření dostává privátní kopii instančních proměnných.*
- *Je-li provedena operace, definovaná pro třídu objektů nad daným objektem, dojde ke změně stavu pouze tohoto objektu.*

# Třídy a objekty - vlastnosti

- **Zapouzdření** (encapsulation) je množina služeb, které objekt nabízí navenek. Odděluje rozhraní (**interface**) a jeho implementaci.
- **Stav** je určen daty objektu.
- **Chování** je určeno stavem objektu a jeho službami (metodami).
- **Identita** je odlišení od ostatních objektů (v prog. jazycích pojmenování proměnných reprezentující objekty určité třídy).

# Struktura objektu

- Objekt je kombinací dat a funkcí, které pracují nad těmito daty

*Funkce procedurálního programování*

- Objekt je tvořen

- **Datovými strukturami** – atributy

- Ovlivňují vlastnosti objektu
- Jsou to proměnné různých datových typů
- Data jsou zpravidla přístupná pouze v rámci daného objektu a zvnějšku jsou skryta před jinými objekty

*Zapouzdření (encapsulation)*

- **Metodami** – funkce / procedury

- Určují chování objektu
- Definují operace nad daty objektu
- Metody představují služby objektu, proto jsou často veřejné

*Mohou být deklarovány jako privátní, např. pro pomocné funkce/výpočtu zlepšující čitelnost kódu.*

# Princip zapouzdření

- „Utajení“ vnitřního stavu objektu
- Jiné objekty nemohou měnit stav objektu přímo a způsobit tak chybu

*Např. konzistence hodnot více proměnných*

- Metody objektu umožňují objektu komunikovat se svým okolím, tvoří jeho **rozhraní**
- Proměnné (data) objektu nejsou z vnějšku objektu přístupné, pro přístup k nim lze využít pouze metody
- Zapouzdření umožňuje udržovat a spravovat každý objekt nezávisle na jiném objektu. Umožňuje **modularitu** zdrojových kódů.

## Komunikace mezi objekty

- V OO systému interagují objekty mezi sebou zasíláním zpráv požadavků na provedení služeb poskytovaných objektem
- Objekty tak mezi sebou komunikují prostřednictvím zpráv, které jsou realizovány (implementovány) metodami
- Pokud jeden objekt požaduje po jiném objektu, aby vykonal nějakou činnost, zašle mu zprávu ve tvaru:
  - **Objekt**, na kterém se má akce provést  
*Referenční proměnná odkazující na objekt, např. String*
  - **Činnost**, která se má vykonat  
*Metoda (procedura, funkce), např. compareTo*
  - **Seznam parametrů** volané metody  
*Parametry funkce*
- Zpráva neobsahuje popis jak činnost vykonat, ale pouze co provést  
*Konkrétní způsob implementace nemusí být dopředu (v průběhu kompilace) znám (viz např. později diskutované virtuální metody).*

# Datové položky třídy a instance

## ■ Datové položky třídy

- Jsou společné všem instancím vytvořeným z jedné třídy
- Nejsou vázaný na konkrétní instanci
- Jsou společné všem instancím třídy
- V Javě jsou uvozeny klíčovým slovem **static**

## ■ Datové položky instance

- Tvoří vlastní sadu datových položek objektu
- Jsou to tzv. proměnné instance
- Jsou iniciovány při vytvoření instance

*V konstruktoru při vytvoření instance voláním `new`*

- Existují po celou dobu života instance
- Proměnné jedné instance jsou **nezávislé** na proměnných instance jiné



# Metody třídy a instance

## ■ Metody třídy

- Nejsou volány pro konkrétní instance
- Představují zprávu zaslanou třídě jako celku
- Mohou pracovat pouze s proměnnými třídy

*Nikoliv s proměnnými instance*

- V Javě jsou uvozeny klíčovým slovem **static**
- Jsou to tzv. statické metody

## ■ Metody instance

- Jsou volány vždy pro konkrétní instanci třídy
- Představují zprávu zaslanou konkrétní instanci
- Pracují s proměnnými instance i s proměnnými třídy
- Lze volat pouze až po vytvoření konkrétní instance

## Přístup ke členům třídy

- Podle principu zapouzdření jsou některé členy třídy označovány jako soukromé (privátní) a jiné jako veřejné.
- Programátor předepisuje k jakým položkám lze přistupovat a modifikovat je
- Přístup ke členům třídy je určen **modifikátorem přístupu**
  - **public:** – přístup z libovolné třídy
  - **private:** – přístup pouze ze třídy, ve které byly deklarovány
  - **protected:** – přístup ze třídy a z odvozených tříd
  - Bez uvedení modifikátoru je přístup povolen v rámci stejného balíčku **package**

## Řízení přístup ke členům třídy

Modifikátor	Přístup			
	Třída	Balíček	Odvozená třída	„Svět“
<b>public</b>	✓	✓	✓	✓
<b>protected</b>	✓	✓	✓	✗
<i>bez modifikátoru</i>	✓	✓	✗	✗
<b>private</b>	✓	✗	✗	✗

<http://docs.oracle.com/javase/tutorial/java/java00/accesscontrol.html>

## Datové položky objektů

- Dle principu zapouzdření jsou datové položky zpravidla **private**
- Přístup k položkám je přes metody, tzv. „accessory“, které vrací/-nastavují hodnotu příslušné proměnné („getter“ a „setter“)

```
public class DemoGetterSetter {  
    private int x;  
}
```

- „Accessory“ lze vytvořit mechanicky a vývojová prostředí zpravidla nabízí automatické vygenerování jejich zdrojového kódu

```
public class DemoGetterSetter {  
    private int x;  
  
    public int getX() {  
        return x;  
    }  
  
    public void setX(int x) {  
        this.x = x;  
    }  
}
```

*Viz Alt+Insert v Netbeans*

## Vytvoření objektu – Konstruktor třídy

- Instance třídy (objekt) vzniká voláním operátoru **new** s argumentem jména třídy, který volá **konstruktor** třídy
- Konstruktor nemá návratový typ, jmenuje se stejně jako třída a můžeme jej přetížit pro různé typy a počty parametrů
- Jiný konstruktor třídy lze volat operátorem **this**

*Operátorem **super** lze volat konstruktor nadřazené třídy*

- Není-li konstruktor předepsán, je vygenerován konstruktor s prázdným seznamem parametrů
  - Je-li konstruktor deklarován, implicitní zaniká
- Přetížení konstruktoru pro různé typy a počty parametrů
- **Konstruktor je zpravidla vždy public** **overloading**
- Privátní (**private**) konstruktor použijeme například pro:
  - Třídy obsahující pouze statické metody nebo pouze konstanty  
*Zakážeme tak vytváření instancí.*
  - Takzvané singletony (singletons)

## Statická metoda **main**

- Deklarace hlavní funkce

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

představuje „spouštěč“ programu

- Musí být statická, je volána dříve než se vytvoří objekt
- Třída nemusí obsahovat funkci **main**
  - Taková třída zavádí prostředky, které lze využít v jiných třídách
  - Jedná se tak o „knihovnu“ funkcí a procedur nebo datových položek (konstant)
- Kromě spuštění programu může funkce **main** obsahovat například testování funkčnosti objektu nebo ukázkou použití metod objektu

## Příklad třídy jako datového typu – třída `Complex`

- Třída `Complex` – představuje třídu datového typu, jejíž objektový návrh a implementace vychází z konceptu zapouzdření
- Datové položky:
  - Hodnoty typu `double` pro reprezentaci reálné a imaginární části (dvojice čísel)
- Metody: tvoří množinu operací obvyklých pro operace nad komplexními čísly
  - absolutní hodnota, sčítání, odčítání, násobení a dělení

**Uvedený příklad je implementací třídy v Javě**

## Třída Complex 1/6

```
public class Complex {  
  
    //data fields  
    private double re = 0.; //data položka (atribut)  
    private double im = 0.; //data položka (atribut)  
    ...  
}
```

- Definice třídy je uvozena klíčovým slovem **class** následovaném jménem třídy
- Kódovací konvence doporučuje psát jméno třídy s prvním písmenem velkým
- Veřejná třída se specifikuje klíčovým slovem (modifikátorem) **public** před **class**
- Datové položky (atributy) se zapisují podobně jako deklarace proměnných

*Kódovací konvence doporučuje zapisovat datové položky jako první*



## Třída Complex 2/6

```
public class Complex {  
    ...  
    public Complex() {}  
    public Complex(double r) {  
        re = r;  
    }  
    public Complex(double r, double i) {  
        re = r;  
        im = i;  
    }  
}
```

- Za datovými položkami následují definice **konstruktoru**(ů)
- Konstruktor je metoda stejného jména jako jméno třídy a nemá návratovou hodnotu
- Konstruktor je volán při vytvoření objektu příkazem **new**, který vrací referenci (odkaz), kde je objekt uložen v paměti

## Třída Complex 3/6

```
public class Complex {  
    ...  
    //methods (operations)  
    public double getAbs() {  
        return Math.sqrt(re * re + im * im);  
    }  
  
    public Complex plus(Complex b) {  
        double r = re + b.re; // r je lokální proměnná  
                               // re je atribut objektu  
        double i = im + b.im;  
        return new Complex(r, i);  
    }  
    ...  
}
```

- Metody jsou funkce s návratovým typem a specifikací přístupových práv

## Třída Complex 4/6

```
public class Complex {  
    ...  
    public Complex minus(Complex b) {  
        Complex a = this;  
        return new Complex(a.re - b.re, a.im - b.im);  
    }  
  
    public Complex times(Complex b) {  
        Complex a = this;  
        double r = a.re * b.re - a.im * b.im;  
        double i = a.re * b.im + a.im * b.re;  
        return new Complex(r, i);  
    }  
}
```

- Uvnitř metody můžeme použít operátor **this**
- **this** je implicitní odkaz na objekt, na který byla metoda zavolána

## Třída Complex 5/6

```
public class Complex {  
    ...  
    public String toString() {  
        if (im == 0) {  
            return re + "";  
        } else if (re == 0) {  
            return im + "i";  
        } else if (im < 0) {  
            return re + " - " + (-im) + "i";  
        }  
        return re + " + " + im + "i";  
    }  
}
```

- **toString** je metoda každého objektu, která vrací řetězec představující znakovou reprezentaci objektu „Dědí od třídy Object“
- Pokud není předefinována vrací jméno třídy + hash kód

*Překrytí je realizováno dynamickou vazbou (polymorfismus)*

## Třída Complex 6/6

```
public class Complex {  
    ...  
    public static Complex plus(Complex a, Complex b) {  
        double r = a.re + b.re;  
        double i = a.im + b.im;  
        Complex sum = new Complex(r, i);  
        return sum;  
    }  
}
```

### Statické metody:

- jsou uvozeny klíčovým slovem **static**
- jsou to metody třídy a nejsou svázány s objektem
- jsou přístupné i bez vytvoření instance třídy (objektu)
- nemají přístup k instančním proměnným (datovým položkám)

*Instanční proměnné se vytvářejí až s vytvořením objektu operátorem new*

## Instance třídy Complex 1/2

```
public static void main(String[] args) {  
    Complex c1 = new Complex(2);  
    Complex c2 = new Complex(2, 1);  
  
    System.out.println("New complex: " + new Complex());  
    System.out.println("Complex var c1: " + c1);  
    System.out.println("Complex var c2: " + c2);  
  
    System.out.println("Complex var |c1|: " + c1.getAbs());  
    System.out.println("Complex var |c2|: " + c2.getAbs());  
  
    System.out.println("Complex var c1-c2: " + c1.minus(c2));  
    System.out.println("Complex var c1+c2: " + c1.plus(c2));  
    System.out.println("Complex var c1*c2: " + c1.times(c2));  
  
    System.out.println("Complex: (1 + j) + (1 - j): " +  
        Complex.plus(new Complex(1, 1), new Complex(1, -1)));  
}
```

- Objekty (instance třídy) Complex vytváříme operátorem **new**

## Instance třídy Complex 2/2

### ■ Příklad výpisu:

```
java DemoComplex

New complex: 0.0
Complex var c1: 2.0
Complex var c2: 2.0 + 1.0i
Complex var |c1|: 2.0
Complex var |c2|: 2.23606797749979
Complex var c1-c2: -1.0i
Complex var c1+c2: 4.0 + 1.0i
Complex var c1*c2: 4.0 + 2.0i
Complex: (1 + j) + (1 - j): 2.0
```

Complex.java a DemoComplex.java

## Přístup k datovým položkám

- Datové položky reprezentující reálnou a komplexní část jsou ve třídě `Complex` skryty.

*Princip zapouzdření*

- Pro přístup k nim, můžeme implementovat metody nazývané

- **getter** – „čtení“

```
public class Complex {  
    ...  
    public double getRe() {  
        return re;  
    }  
    public double getIm() {  
        return im;  
    }  
    ...  
}
```

- **setter** – „zápis“

```
public class Complex {  
    ...  
    public void setRe(double re) {  
        this.re = re;  
    }  
    public void setIm(double im) {  
        this.im = im;  
    }  
    ...  
}
```

Jakou má výhodu přistupovat k proměnným přes metody?



# Neměnitelné objekty (Immutable objects)

## ■ Definice neměnitelného objektu

- Všechny datové položky jsou **final** a **private**
- Neimplementujeme „settery“ pro modifikaci položek
- Zákaz přepisu metod v potomcích (**final** modifikátor u metod)

*Nebo jednoduše zákaz odvozování uvedením **final class***

<http://docs.oracle.com/javase/tutorial/essential/concurrency/imstrat.html>

## ■ Objekty, které v průběhu života **nemění svůj stav**

- Modifikace objektu není možná a je nutné vytvořit objekt nový
- Mají výhodu v případě paralelního běhu více výpočetních toků

*Typickým příkladem je instance třídy **String***

<http://docs.oracle.com/javase/tutorial/essential/concurrency/immutable.html>

## Příklad – **final** nezaručuje neměnitelnost objektu 1/3

- Definujme **final** třídu zapouzdřující referenční **private final** proměnnou **values** typu odkaz na pole **int** hodnot
- Přístup k proměnné **values** je pouze přes *getter* **getArray()**

```
public final class ArrayWrapper {  
    private final int[] values;  
  
    public ArrayWrapper(int n) {  
        values = new int[n];  
  
        for (int i = 0; i < values.length; i++) {  
            values[i] = (int) (Math.random() * n);  
        }  
    }  
  
    public int[] getArray() {  
        return values;  
    }  
  
    public String toString() {  
        ...  
    }  
}
```

ArrayWrapper.java

## Příklad – **final** nezaručuje neměnitelnost objektu 2/3

- Obsah objektu `ArrayWrapper` vypíšeme metodou `toString()`

```
public final class ArrayWrapper {  
    ...  
    public String toString() {  
        ...  
        StringBuilder sb = new StringBuilder(values.length >  
0 ? new Integer(values[0]).toString() : "empty");  
        for (int i = 1; i < values.length; i++) {  
            sb.append(" ");  
            sb.append(values[i]);  
        }  
        return sb.toString();  
    }  
}
```

`ArrayWrapper.java`

## Příklad – **final** nezaručuje neměnitelnost objektu 3/3

```
final ArrayWrapper a = new ArrayWrapper(10);
```

```
System.out.println("Random final array '" + a + "'");
```

- Po vytvoření objektu `ArrayWrapper` je obsah pole inicializován v konstruktoru na náhodná čísla

```
Random final array '1 5 5 9 4 9 7 1 8 1'
```

- Přístup k položkám pole máme přes *getter* `getArray()`

```
final int[] v = a.getArray();
```

```
for (int i = 0; i < v.length; i++) {
```

```
    v[i] = i;
```

```
}
```

```
System.out.println("Final object and final array can be  
still modified\n'" + a + "'");
```

- Původní „final” objekt tak můžeme změnit

```
Final object and final array can be still modified
```

```
'0 1 2 3 4 5 6 7 8 9'
```

`FinalArrayDemo.java`

Referenční datové položky neměnitelných objektů musí odkazovat na **neměnitelné objekty**.

## Objekty pro základní typy

- Třídy pro základní třídy jsou immutable
- Každý primitivní typ má v Javě také svoji třídu:
  - **Char**, **Boolean**
  - **Byte**, **Short**, **Integer**, **Long**
  - **Float**, **Double**
- Třídy obsahují metody pro převod čísel a metody pro parsování čísla z textového řetězce
  - např. **Integer.parseInt(String s)**
- Dále také rozsah číselného typu minimální a maximální hodnoty
  - např. **Integer.MAX\_VALUE**, **Integer.MIN\_VALUE**

<http://docs.oracle.com/javase/8/docs/api/java/lang/Number.html>

## Referenční proměnné objektů tříd primitivních typů

- Referenční proměnné objektů pro primitivní typy můžeme používat podobně jako základní typy

```
Integer a = 10;  
Integer b = 20;  
  
int r1 = a + b;  
Integer r2 = a + b;  
System.out.println("r1: " + r1 + " r2: " + r2);
```

- Stále to jsou však referenční proměnné (odkazující na adresu v paměti)
- Obsah objektu nemůžeme měnit, jedná se o **immutable** objekty

DemoObjectsOfBasicTypes.java

# Shrnutí přednášky

# Diskutovaná témata

- Informace o předmětu
- Úvod do programovacího jazyku Java
- Třídy, objekty a objektově orientované programování
- Vlastnosti objektů a tříd, zapouzdření
- Neměnitelné (immutable) objekty
  
- Příště: Objekty, vztahy, vlastnosti, dědičnost a polymorfismus, ...