Programovací jazyk Java

zaujímavosti a zopakovanie

Andrej Ferko

4.10.2007

3 piliere OOP

- Encapsulation
- Inheritance
- Polymorphism

3 piliere OOP

- Encapsulation

  private field + getter/setter

- Inheritance

  extends

- Polymorphism

  compile time / runtime

Primitive data types

Int division

Čo vypíše program?

```
public class Division {
  public static void main(String[] args) {
    int i = 5;
    System.out.println(i / 2);
    System.out.println(i / 2F);
  }
}
```

Primitive data types

Int division

Čo vypíše program?

```
public class Division {
  public static void main(String[] args) {
    int i = 5;
    System.out.println(i / 2);
    System.out.println(i / 2F);
  }
}
```

2
2.5

- typ výsledku pri aritmetických operáciach
- prehlásenie čísla za float suffixom F/f, double suffixom D/d, long suffixom L/l

Primitive data types

It’s elementary

Čo vypíše program?

a) -22430
b) 59753
c) 10864
d) 108642

```
public class Elementary {
  public static void main(String[] args) {
    System.out.println(54321 + 5432l);
  }
}
```

Primitive data types

It’s elementary

Čo vypíše program?
a) -22430
b) 59753
c) 10864
d) 108642

```
public class Elementary {
  public static void main(String[] args) {
    System.out.println(54321 + 5432l);
  }
}
```

b)

- písmeno l sa pletie s jednotkou (písmeno l nie je ani vhodný názov premennej)
- radšej teda `System.out.println(54321 + 5432L);`

Primitive data types

Auto upcast

Čo vypíšu jednotlivé riadky?

```
int a = 4;

System.out.println(a < 5 ? "1" : new Integer(2));



System.out.println(a < 5 ? 1 : 2.3);
```

Primitive data types

Auto upcast

Čo vypíšu jednotlivé riadky?

```
int a = 4;

System.out.println(a < 5 ? "1" : new Integer(2));
```

kompilačná chyba:
Incompatible conditional operand types String and Integer

```
System.out.println(a < 5 ? 1 : 2.3);
```

1.0
pre int nastal automaticky upcast na double

Primitive data types

Long division

Nasledujúci program robí jednoduché delenie. Čo sa stane ked sa program spustí?

a) vypíše sa 5
b) vypíše sa 1000
c) vypíše sa 5000
d) vyhodí výnimku

```
public class Division {
  public static final long
        MILISEKUNDY_ZA_DEN = 24 * 60 * 60 * 1000;
  public static final long
        MIKROSEKUNDY_ZA_DEN = 24 * 60 * 60 * 1000 * 1000;
  public static void main(String[] args) {
    System.out.println(MIKROSEKUNDY_ZA_DEN / MILISEKUNDY_ZA_DEN);
  }
}
```

Primitive data types

Long division

Nasledujúci program robí jednoduché delenie. Čo sa stane ked sa program spustí?
a) vypíše sa 5
b) vypíše sa 1000
c) vypíše sa 5000
d) vyhodí výnimku

```
public class Division {
  public static final long
        MILISEKUNDY_ZA_DEN = 24 * 60 * 60 * 1000;
  public static final long
        MIKROSEKUNDY_ZA_DEN = 24 * 60 * 60 * 1000 * 1000;
  public static void main(String[] args) {
    System.out.println(MIKROSEKUNDY_ZA_DEN / MILISEKUNDY_ZA_DEN);
  }
}
```

a)

- rozsah primitívnych typov
- vyhodnocovanie operácií zľava doprava (najprv podľa zátvoriek a priority operátorov)

Primitive data types

Smart sort (or not?)

Program ma usporiadať čísla podľa veľkosti. Je v poriadku?

```
public class Sort {
  public static void main(String args[]) {
    Integer big = new Integer(2000000000);
    Integer small = new Integer(-2000000000);
    Integer zero = new Integer(0);
    Integer[] a = new Integer[] { big, small, zero };
    java.util.Arrays.sort(a, new java.util.Comparator() {
      public int compare(Object o1, Object o2) {
        return ((Integer) o2).intValue() - ((Integer) o1).intValue();
      }
    });
    System.out.println(java.util.Arrays.asList(a));
  }
}
```

Primitive data types

Smart sort (or not?)

Program ma usporiadať čísla podľa veľkosti. Je v poriadku?

```
public class Sort {
  public static void main(String args[]) {
    Integer big = new Integer(2000000000);
    Integer small = new Integer(-2000000000);
    Integer zero = new Integer(0);
    Integer[] a = new Integer[] { big, small, zero };
    java.util.Arrays.sort(a, new java.util.Comparator() {
      public int compare(Object o1, Object o2) {
        return ((Integer) o2).intValue() - ((Integer) o1).intValue();
      }
    });
    System.out.println(java.util.Arrays.asList(a));
  }
}
```

Comparator je zlý.
Dátovy typ int ma malý rozsah pre výsledok.

Primitive data types

Magic float

Čo vypíše nasledujúci program?

```
public class FloatTest {
  public static void main(String[] args) {
    float f = 2000000000;
    System.out.println((f + 64) - f);
    System.out.println((f + 65) - f);
  }
}
```

Primitive data types

Magic float

Čo vypíše nasledujúci program?

```
public class FloatTest {
  public static void main(String[] args) {
    float f = 2000000000;
    System.out.println((f + 64) - f);
    System.out.println((f + 65) - f);
  }
}
```

0.0
128.0

- aritmetika reálnych čísel

Primitive data types

Down for the count

Čo vypíše nasledujúci program?

a) 0

b) 50

c) 51

d) nič z uvedeného

```
public class Count {
  public static void main(String[] args) {
    final int START = 2000000000;
    int count = 0;
    for (float f = START; f < START + 50; f++)
      count++;
    System.out.println(count);
  }
}
```

Primitive data types

Down for the count

Čo vypíše nasledujúci program?
a) 0
b) 50
c) 51
d) nič z uvedeného

```
public class Count {
  public static void main(String[] args) {
    final int START = 2000000000;
    int count = 0;
    for (float f = START; f < START + 50; f++)
      count++;
    System.out.println(count);
  }
}
```

a)

- aritmetika reálnych čísel
- pre presné výpočty nepoužívame float a double ale java.math.BigDecimal

Primitive data types

Keyword	Description	Size/Format
(integers)		
byte	Byte-length integer	8-bit two's complement
short	Short integer	16-bit two's complement
int	Integer	32-bit two's complement
long	Long integer	64-bit two's complement
(real numbers)		
float	Single-precision floating point	32-bit IEEE 754
double	Double-precision floating point	64-bit IEEE 754
(other types)		
char	A single character	16-bit Unicode character
boolean	A boolean value (true or false)	true or false

Primitive data types

Examples of literal values

Literal	Data Type
178	int
8864L	long
37.266	double
37.266D	double
87.363F	float
26.77e3	double
'c'	char
true	boolean
false	boolean

Primitive data types

char

- číselný unsigned type
- V Jave každý znak v pamäti (teda char) má 16 bitov.
- Rozdiel medzi znakovou sadou Unicode a jej kódovaniami:
  - Unicode - abstraktná množina znakov s poradovými číslami
  - kódovanie - zápis znakovej sady pomocou bitov
    - UCS-4, UTF-8, UTF-16, MojeVlastneKodovanie, ...
- Editory (napr. Notepad) ponúkajú uloženie v Unicode, ale takéto kódovanie neexistuje. (Notepad tým myslí pravdepodobne UTF-16.)

Primitive data types

Čo je výsledkom nasledujúceho programu?
a) vypíše sa 11 krát 100
b) vypíše sa 10 krát 100 a nastane runtime exception
c) kompilačná chyba - premenná i nemôže byť deklarovaná dvakát v rámci main
d) kompilačná chyba - premenná j nemôže byť deklarovaná dvakát v rámci switch
e) kompilačná chyba - riadiaca premenná vo switch nemôže byť typu char
f) nič z uvedeného

```
public static void main(String args[]) {
  char digit = 'a';
  for (int i = 0; i < 10; i++) {
    switch (digit) {
      case 'x': {
        int j = 0;
        System.out.println(j);
      }
      default: {
        int j = 100;
        System.out.println(j);
      }
    }
  }
  int i = j;
  System.out.println(i);
}
```

Primitive data types

Čo je výsledkom nasledujúceho programu?
a) vypíše sa 11 krát 100
b) vypíše sa 10 krát 100 a nastane runtime exception
c) kompilačná chyba - premenná i nemôže byť deklarovaná dvakát v rámci main
d) kompilačná chyba - premenná j nemôže byť deklarovaná dvakát v rámci switch
e) kompilačná chyba - riadiaca premenná vo switch nemôže byť typu char
f) nič z uvedeného

```
public static void main(String args[]) {
  char digit = 'a';
  for (int i = 0; i < 10; i++) {
    switch (digit) {
      case 'x': {
        int j = 0;
        System.out.println(j);
      }
      default: {
        int j = 100;
        System.out.println(j);
      }
    }
  }
  int i = j;
  System.out.println(i);
}
```

f)
kompilačná chyba na riadku
`int i = j;`
za cyklom for, pretože žiadna premenná j v danom mieste nie je viditeľná

Primitive data types

boolean

Ako sprehľadniť nasledujúci program?

```
if (go == true) {
  System.out.println("go");
  System.out.println("on");
} else
  return;
```

Primitive data types

boolean

Ako sprehľadniť nasledujúci program?

```
if (go == true) {
  System.out.println("go");
  System.out.println("on");
} else
  return;
```

Napr. takto:

```
if (!go)
  return;

System.out.println("go");
System.out.println("on");
```

- pri porovnaniach boolean premennej vzhľadom k **`true`**/**`false`** je použitie == zbytočné
- opustenie konštrukcie (tu cez **`return`**) sa dá často ošetriť na začiatku, čo sprehľadní kód

Reference data types

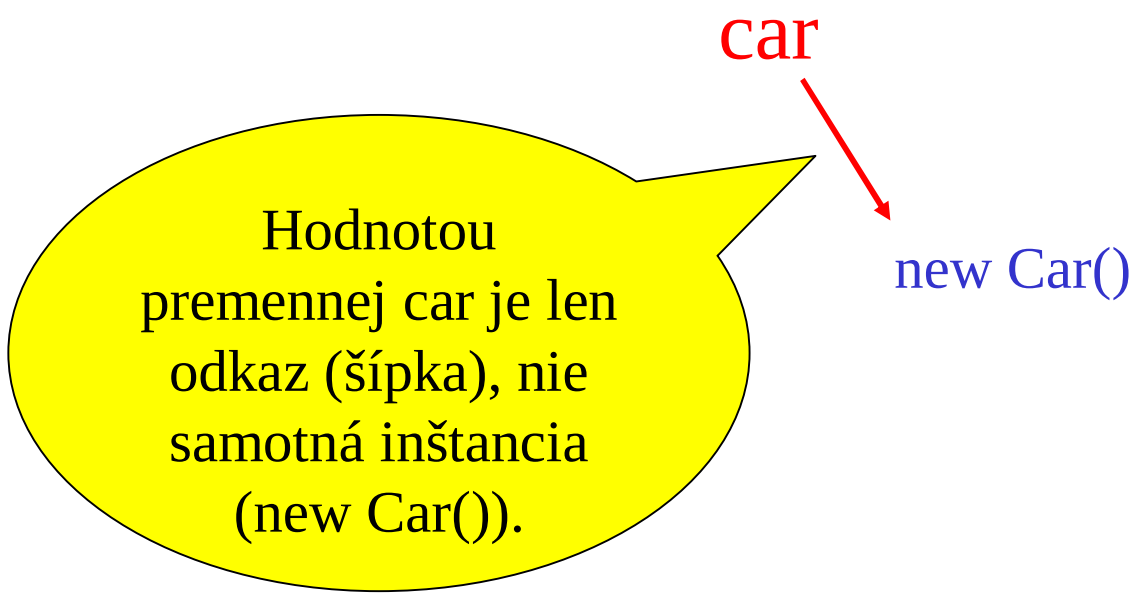
- Arrays, Classes, Interfaces
- hodnotou reference data type premennej je odkaz na hodnotu reprezentovanú premennou

int i = 5;

i

5

Car car = new Car();



•Stack

•Heap (garbage collector)

Reference data types

Parametre metód

V programovaní rozlišujeme odovzdávanie parametrov metódam:

pass-by-value

- Parameter je do metódy skopírovaný zo skutočného parametra.
- V metóde sa používa len kópia parametra.

pass-by-reference

- Parameter v metóde je použitý len ako alias na skutočný parameter.
- V metóde sa používa skutočný parameter.

Java je pass-by-value.

Reference data types

Java je pass-by-value

Pre primitive data types je to jasné.

```
public class Primitive {

  public static int square(int i) {
    i = i * i;
    return i;
  }

  public static void main(String[] args) throws Exception {
    int i = 5;
    System.out.println(i);
    System.out.println(square(i));
    System.out.println(i);
  }
}
```

Reference data types

Java je pass-by-value

Pre primitive data types je to jasné.

```
public class Primitive {

  public static int square(int i) {
    i = i * i;
    return i;
  }

  public static void main(String[] args) throws Exception {
    int i = 5;
    System.out.println(i);
    System.out.println(square(i));
    System.out.println(i);
  }
}
```

```
5
25
5
```

Reference data types

Java je pass-by-value

Prečo sú reference data types pass-by-value?

Jednoduchý test či jazyk podporuje odovzdávanie parametrov metódam pass-by-reference:

Da sa napísať metóda swap(a,b)?

Reference data types

swap(a,b) v Pascale

```
{ Pascal }
procedure swap(var arg1, arg2: SomeType);
  var
    temp : SomeType;
  begin
    temp := arg1;
    arg1 := arg2;
    arg2 := temp;
  end;


...
{ in some other procedure/function/program }
var
  var1, var2 : SomeType;
begin
  var1 := ...;
  var2 := ...;
  swap(var1, var2);
end;
```

Reference data types

swap(a,b) v C++

```
// C++
void swap(SomeType& arg1, Sometype& arg2) {
  SomeType temp = arg1;
  arg1 = arg2;
  arg2 = temp;
}

...

SomeType var1 = ...;
SomeType var2 = ...;
swap(var1, var2); // swaps their values!
```

Reference data types

Java je pass-by-value

V Jave swap(a,b) nenapíšeme.

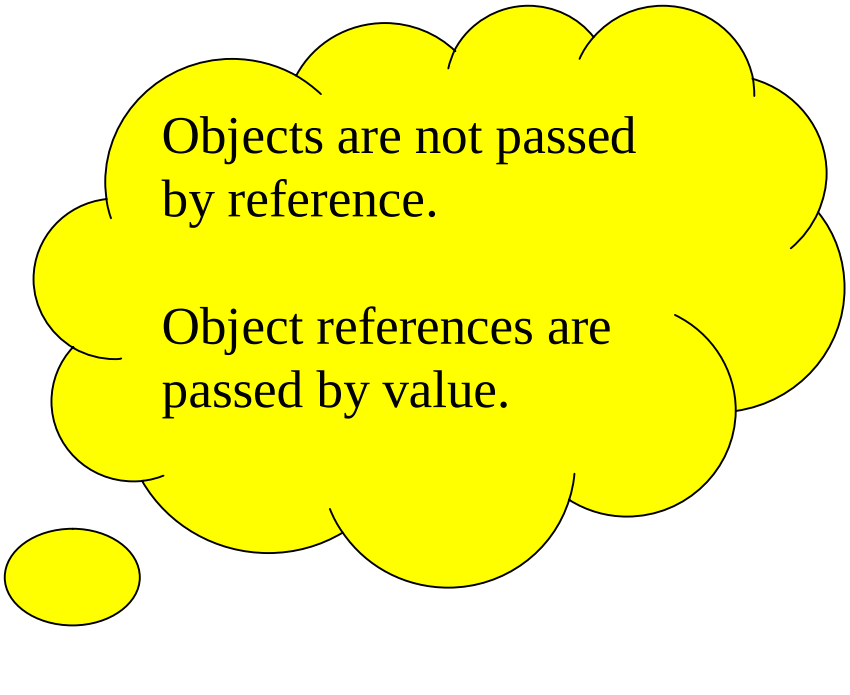
```
public void foo(Dog d) {
  d = new Dog("Fifi");
}

Dog aDog = new Dog("Max");
foo(aDog);
```

Premenná `aDog` aj po volaní metódy `foo` stále ukazuje na „Max“.

Reference data types

Java je pass-by-value



Mnohí chybne predpokladajú, že pri volaní

```
public void foo(Dog d) {
  d.setName("Fifi");
}
```

sa inštancia `d` odovzdá by-reference.

`Dog d` však nepredstavuje priamo inštanciu, ale len odkaz na ňu.

Reference data types

```
class Value {
  public int i = 15;
}

public class Test {

  public static void main(String argv[]) {
    Test t = new Test();
    int i = 5;
    Value v = new Value();
    v.i = 25;
    t.method(v, i);
    System.out.println(v.i);
  }

  public void method(Value v, int i) {
    i = 0;
    v.i = 20;
    Value val = new Value();
    v = val;
    System.out.println(v.i + " " + i);
  }
}
```

Čo vypíše tento program?

Reference data types

```
class Value {
  public int i = 15;
}

public class Test {

  public static void main(String argv[]) {
    Test t = new Test();
    int i = 5;
    Value v = new Value();
    v.i = 25;
    t.method(v, i);
    System.out.println(v.i);
  }

  public void method(Value v, int i) {
    i = 0;
    v.i = 20;
    Value val = new Value();
    v = val;
    System.out.println(v.i + " " + i);
  }
}
```

Čo vypíše tento program?

15 0
20

Reference data types

null

Čo vypíšu jednotlivé riadky?

```
Integer i = null;
String s = null;

System.out.println(i instanceof Number);


System.out.println(null + "X");


System.out.println(null + i);


System.out.println(i + s);


System.out.println(i + i + s);


System.out.println(s + null);


System.out.println(null + s);
```

Reference data types

null

Čo vypíšu jednotlivé riadky?

```
Integer i = null;
String s = null;

System.out.println(i instanceof Number);  false


System.out.println(null + "X");           nullX


System.out.println(null + i);             kompilačná chyba:
  The operator + is undefined for the argument type(s) null, Integer

System.out.println(i + s);                nullnull


System.out.println(i + i + s);            kompilačná chyba:
  The operator + is undefined for the argument type(s) Integer, Integer

System.out.println(s + null);             nullnull


System.out.println(null + s);             nullnull
```

Interfaces, class hierarchy and cast

Predpokladajme že máme takéto triedy a sekvenciu príkazov:

```
class FourWheeler implements DrivingUtilities
class Car extends FourWheeler
class Truck extends FourWheeler
class Bus extends FourWheeler
class Crane extends FourWheeler
```

```
1 DrivingUtilities du;
2 FourWheeler fw;
3 Truck myTruck = new Truck();
4 du = (DrivingUtilities) myTruck;
5 fw = new Crane();
6 fw = du;
```

Ktoré výroky sú pravdivé?

a) Na riadku 4 je kompilačná chyba, pretože interface sa nemôže odkazovať na objekt.
b) Na riadku 4 je kompilačná chyba, pretože objekt nemôžeme precastovať na interface.
c) Na riadku 4 je kompilačná chyba, pretože cast tam nie je potrebný.
d) Na riadku 4 nie je cast potrebný, ale nie je to kompilačná chyba.
e) Na riadku 6 je kompilačná chyba, pretože je potrebný explicitný cast.
f) Ak na riadku 6 doplníme cast, tak sa riadok skompiluje, ale skončí chybou v runtime.
g) V programe nie je žiadna kompilačná chyba.
h) V programe je nejaká kompilačná chyba.

Interfaces, class hierarchy and cast

Predpokladajme že máme takéto triedy a sekvenciu príkazov:

```
class FourWheeler implements DrivingUtilities
class Car extends FourWheeler
class Truck extends FourWheeler
class Bus extends FourWheeler
class Crane extends FourWheeler
```

```
1 DrivingUtilities du;
2 FourWheeler fw;
3 Truck myTruck = new Truck();
4 du = (DrivingUtilities) myTruck;
5 fw = new Crane();
6 fw = du;
```

Ktoré výroky sú pravdivé?

a) Na riadku 4 je kompilačná chyba, pretože interface sa nemôže odkazovať na objekt.

b) Na riadku 4 je kompilačná chyba, pretože objekt nemôžeme precastovať na interface.

c) Na riadku 4 je kompilačná chyba, pretože cast tam nie je potrebný.

d) Na riadku 4 nie je cast potrebný, ale nie je to kompilačná chyba.

e) Na riadku 6 je kompilačná chyba, pretože je potrebný explicitný cast.

f) Ak na riadku 6 doplníme cast, tak sa riadok skompiluje, ale skončí chybou v runtime.

g) V programe nie je žiadna kompilačná chyba.

h) V programe je nejaká kompilačná chyba.

Methods and fields

Polymorfizmus

```
class Car {
  public String getName() {
    return "car";
  }
  public void info() {
    System.out.println(getName());
  }
}

class Trabant extends Car {
  public String getName() {
    return "Trabant";
  }
}

public class CarTest {
  public static void main(String args[]) {
    Car car = new Car();
    Car trabant = new Trabant();
    car.info();
    trabant.info();
  }
}
```

Methods and fields

Polymorfizmus

```
class Car {
  public String getName() {
    return "car";
  }
  public void info() {
    System.out.println(getName());
  }
}

class Trabant extends Car {
  public String getName() {
    return "Trabant";
  }
}

public class CarTest {
  public static void main(String args[]) {
    Car car = new Car();
    Car trabant = new Trabant();
    car.info();
    trabant.info();
  }
}
```

car
Trabant

Methods and fields

Instance methods and fields

```
class Car {
  public String name = "car";
  public void info() {
    System.out.println("I'm a car.");
  }
}

class Trabant extends Car {
  public String name = "Trabant";
  public void info() {
    System.out.println("I'm Trabant.");
  }
}

public class CarTest {
  public static void main(String args[]) {
    Car car = new Car();
    Car trabant = new Trabant();
    System.out.println(car.name);
    System.out.println(trabant.name);
    car.info();
    trabant.info();
  }
}
```

Methods and fields

Instance methods and fields

```
class Car {
  public String name = "car";
  public void info() {
    System.out.println("I'm a car.");
  }
}

class Trabant extends Car {
  public String name = "Trabant";
  public void info() {
    System.out.println("I'm Trabant.");
  }
}

public class CarTest {
  public static void main(String args[]) {
    Car car = new Car();
    Car trabant = new Trabant();
    System.out.println(car.name);
    System.out.println(trabant.name);
    car.info();
    trabant.info();
  }
}
```

```
car
car
I'm a car.
I'm Trabant.
```

Methods and fields

Static methods and fields

```
class Car {
  public static String name = "car";
  public static void info() {
    System.out.println("I'm a car.");
  }
}

class Trabant extends Car {
  public static String name = "Trabant";
  public static void info() {
    System.out.println("I'm Trabant.");
  }
}

public class CarTest {
  public static void main(String args[]) {
    Car car = new Car();
    Car trabant = new Trabant();
    System.out.println(car.name);
    System.out.println(trabant.name);
    car.info();
    trabant.info();
  }
}
```

Methods and fields

Static methods and fields

```
class Car {
  public static String name = "car";
  public static void info() {
    System.out.println("I'm a car.");
  }
}

class Trabant extends Car {
  public static String name = "Trabant";
  public static void info() {
    System.out.println("I'm Trabant.");
  }
}

public class CarTest {
  public static void main(String args[]) {
    Car car = new Car();
    Car trabant = new Trabant();
    System.out.println(car.name);
    System.out.println(trabant.name);
    car.info();
    trabant.info();
  }
}
```

```
car
car
I'm a car.
I'm a car.
```

- statické metódy nemôžu byť prekryté, patria triede
- nevolajme ich na inštanciách, len na triedach

Správne volanie:

```
Car.info();
Trabant.info();
```

Keywords

Čo vypíše nasledujúci program?

```
public class A {
  public int do(int i) {return(1);}
  public int do(int i, int j) {return(2);}
  public static void main(String [] args) {
    A a = new A();
    System.err.println(a.do(3));
  }
}
```

Keywords

Čo vypíše nasledujúci program?

```
public class A {
  public int do(int i) {return(1);}
  public int do(int i, int j) {return(2);}
  public static void main(String [] args) {
    A a = new A();
    System.err.println(a.do(3));
  }
}
```

Program sa neskompiluje pretože **do** je kľúčové slovo.

```
do {
  ...
} while (condition);
```

Keywords

Ktoré z deklarácii sú správne?

a) int int;

b) int String;

c) String int;

d) String String;

Keywords

Ktoré z deklarácii sú správne?
a) int int;
b) int String;
c) String int;
d) String String;

Keďže `int` je kľúčové slovo a String nie, tak b) a d).

Conditions

Dané sú podmienky:

A:
```
if (str != null && str.length() >= 3)
  System.out.println("dlhy");
```

B:
```
if (str.length() < 3 || str == null)
  System.out.println("kratky");
```

Čo sa vypíše na výstup pri podmienke:
a) A ak str je `null`
b) A ak str je `"aha"`
c) B ak str je `null`
d) B ak str je `"aha"`

Conditions

Dané sú podmienky:

A:
```
if (str != null && str.length() >= 3)
  System.out.println("dlhy");
```

B:
```
if (str.length() < 3 || str == null)
  System.out.println("kratky");
```

Čo sa vypíše na výstup pri podmienke:
a) A ak str je `null`
b) A ak str je `"aha"`
c) B ak str je `null`
d) B ak str je `"aha"`

a) nevypíše sa nič
b) `"dlhy"`
c) nastane `java.lang.NullPointerException`
d) nevypíše sa nič
oprava podmienky B: `if (str == null || str.length() < 3)`

if

Čo vypíše program?

```
boolean a = true;
boolean b = false;
boolean c = true;
if (a == true)
if (b == true)
if (c == true) System.out.println(1);
else System.out.println(2);
else if (a && (b = c)) System.out.println(3);
else System.out.println(4);
```

if

Pomôžeme si najprv formátovaním kódu.

```
boolean a = true;
boolean b = false;
boolean c = true;
if (a == true)
  if (b == true)
    if (c == true)
      System.out.println(1);
    else
      System.out.println(2);
  else if (a && (b = c))
    System.out.println(3);
  else
    System.out.println(4);
```

if

Pomôžeme si najprv formátovaním kódu.

```
boolean a = true;
boolean b = false;
boolean c = true;
if (a == true)
  if (b == true)
    if (c == true)
      System.out.println(1);
    else
      System.out.println(2);
  else if (a && (b = c))
    System.out.println(3);
  else
    System.out.println(4);
```

3

- formátovanie kódu program výrazne sprehľadní
- uvedené formátovanie nie je optimálne, zišli by sa aj zátvorky a to aj v prípade jednopríkazových vetiev príkazu if

switch

Aká bude hodnota premennej `i` po skončení príkazu **`switch`**?

```
int i = 0;
switch (i) {
  case 0:
    i = 1;
  case 2:
  case 3:
    i = 2;
}
```

switch

Aká bude hodnota premennej `i` po skončení príkazu **`switch`**?

```
int i = 0;
switch (i) {
  case 0:
    i = 1;
  case 2:
  case 3:
    i = 2;
}
```

2

- nezabúdajme na príkaz **`break`** v rámci príkazu **`switch`**
- v prípade nájdenia vyhovujúceho **`case`** program beží ďalej od začiatku nájdenej **`case`** vetvy aj cez nasledujúce **`case`** vetvy, funguje akoby goto príkaz

```
int i = 0;
switch (i) {
  case 0:
    i = 1;
    break;
  case 2:
  case 3:
    i = 2;
}
```

switch

Pri príkaze **`switch`** je vhodné:

- vždy mať aj **`default`**, hoci len taký ktorý vyhodí výnimku
- v každej možnosti ktorá nekončí na **`return`** alebo **`throw`** dávať **`break`**
- ak niekde zámerne **`break`** nedávame, tak namiesto break dať komentár

break and continue with label

V dobre odôvodnených prípadoch je použitie **`break`**/**`continue`** s návestím najčitateľnejsia forma vyskočenia z vnorených konštrukcií.

```
for_i:
for (int i = 0; i < arr.length; i++) {
  switch (arr[i]) {
    case 0:
      tmp = 0;
      break;
    case 1:
    case 2:
      tmp = 1;
      break for_i;
    default:
      tmp = 10;
      break;
  }
}
```

```
for_i:
for (int i = 0, k = 0; i < arr.length; i++) {
  for (int j = 0; j < k; j++) {
    if (tmp[j] == arr[i])
      continue for_i;
  }
  tmp[k++] = arr[i];
}
```

Source code

Line printer

Čo vypíše nasledujúci program?
a) dva prázdne riadky
b) 10
c) neskompiluje sa
d) ako kedy a ako kde

```
public class LinePrinter {
  public static void main(String[] args) {
    // Note: \u000A is Unicode newline
    char c = 0x000A;
    System.out.println(c);
  }
}
```

Source code

Line printer

Čo vypíše nasledujúci program?
a) dva prázdne riadky
b) 10
c) neskompiluje sa
d) ako kedy a ako kde

```
public class LinePrinter {
  public static void main(String[] args) {
    // Note: \u000A is Unicode newline
    char c = 0x000A;
    System.out.println(c);
  }
}
```

c)

- kdekoľvek v zdrojovom kóde v Jave môžeme použiť namiesto znaku zápis pomocou jeho kódu v tvare \uXXXX

Arrays

Je v programe nejaká kompilačná alebo runtimová chyba?
Ak nie, tak čo vypíše?

```
Test1[] t1 = new Test1[10];
Test1[][] t2 = new Test1[5][];
if (t1[0] == null) {
  t2[0] = new Test1[10];
  t2[1] = new Test1[10];
  t2[2] = new Test1[10];
  t2[3] = new Test1[10];
  t2[4] = new Test1[10];
}
System.out.println(t1[0]);
System.out.println(t2[1][0]);
```

Arrays

Je v programe nejaká kompilačná alebo runtimová chyba?
Ak nie, tak čo vypíše?

```
Test1[] t1 = new Test1[10];
Test1[][] t2 = new Test1[5][];
if (t1[0] == null) {
  t2[0] = new Test1[10];
  t2[1] = new Test1[10];
  t2[2] = new Test1[10];
  t2[3] = new Test1[10];
  t2[4] = new Test1[10];
}
System.out.println(t1[0]);
System.out.println(t2[1][0]);
```

null
null

Collections

- java.util.List
- java.util.Set
- java.util.Map

Pri deklarácii používame interface a nie konkrétny typ:

```
java.util.ArrayList list = new java.util.ArrayList();
```

```
java.util.List list = new java.util.ArrayList();
```

Rovnako sa snažíme používať ako parameter metódy interface a nie konkrétny typ.

Collections

iterácia

- java.util.Iterator

```
for (Iterator i = col.iterator(); i.hasNext();) {
  Object colElement = i.next();
  System.out.println(colElement);
}
```

- for/in od Java 5.0

```
for (Object colElement : col) {
  System.out.println(colElement);
}
```

- okrem iterátora, ktorý majú všetky Collections sa špecifické Collections dajú iterovať aj inak, napr. List štandardným `for` cez `list.get(int)`

Collections

Collection -> array

```
col.toArray(new String[col.size()])
```

Collections

array -> List

Čo vypíše nasledujúci program?

```
public class ArrayToList {
  public static void main(String[] args) {
    String[] s = new String[] { "a", "b" };
    java.util.List list = java.util.Arrays.asList(s);
    list.add("c");
    System.out.println(list);
  }
}
```

Collections

array -> List

Čo vypíše nasledujúci program?

```
public class ArrayToList {
  public static void main(String[] args) {
    String[] s = new String[] { "a", "b" };
    java.util.List list = java.util.Arrays.asList(s);
    list.add("c");
    System.out.println(list);
  }
}
```

Po spustení chyba na riadku:

```
list.add("c");
```

```
java.lang.UnsupportedOperationException
at java.util.AbstractList.add(AbstractList.java:150)
at java.util.AbstractList.add(AbstractList.java:88)
at ArrayToList.main(ArrayToList.java:5)
Exception in thread "main"
```

Collections

array -> List

Pozrieme do Javadoc

```
public class ArrayToList {
  public static void main(String[] args) {
    String[] s = new String[] { "a", "b" };
    java.util.List list = java.util.Arrays.asList(s);
    list.add("c");
    System.out.println(list);
  }
}
```

asList
public static List **asList**(Object[] a)
Returns a fixed-size list backed by the specified array. ...

Ak chceme meniť počet prvkov v zozname, musíme použiť:

```
java.util.List list = new java.util.ArrayList(java.util.Arrays.asList(s));
```

Collections

hashCode()

Čo vypíše nasledujúci program?

```
public class Name {
  private String first, last;

  public Name(String first, String last) {
    this.first = first;
    this.last = last;
  }

  public boolean equals(Object o) {
    if (!(o instanceof Name))
      return false;
    Name n = (Name) o;
    return n.first.equals(first) && n.last.equals(last);
  }

  public static void main(String[] args) {
    java.util.Set s = new java.util.HashSet();
    s.add(new Name("John", "Lennon"));
    System.out.println(s.contains(new Name("John", "Lennon")));
  }
}
```

Collections

hashCode()

- výsledok je nedefinovaný, pretože sme porušili kontrakt metódy hashCode()
- pri prekrývaní metódy equals(Object), treba prekryť aj metódu hashCode()
- inak nastávajú problémy pri používaní HashMap, HashSet

String

Swimming Strings

Čo vypíše nasledujúci program?

```
public class StringPool {
  static String s1 = "I am unique!";
  public static void main(String args[]) {
    String s2 = "I am unique!";
    String s3 = new String(s1);
    System.out.println(s1 == s2);
    System.out.println(s1.equals(s2));
    System.out.println(s3 == s1);
    System.out.println(s3.equals(s1));
    System.out.println(Another.s4 == s1);
  }
}

class Another {
  static String s4 = "I am unique!";
}
```

String

Swimming Strings

Čo vypíše nasledujúci program?

```
public class StringPool {
  static String s1 = "I am unique!";
  public static void main(String args[]) {
    String s2 = "I am unique!";
    String s3 = new String(s1);
    System.out.println(s1 == s2);
    System.out.println(s1.equals(s2));
    System.out.println(s3 == s1);
    System.out.println(s3.equals(s1));
    System.out.println(Another.s4 == s1);
  }
}

class Another {
  static String s4 = "I am unique!";
}
```

true
true
false
true
true

- pool Stringov

String

The name game

Čo vypíše nasledujúci program?

a) 0

b) 1

c) 2

d) rôzny výsledok pri rôznom spustení

```
public class NameGame {
  public static void main(String[] args) {
    Map m = new IdentityHashMap();
    m.put("Mickey", "Mouse");
    m.put("Mickey", "Mantle");
    System.out.println(m.size());
  }
}
```

String

The name game

Čo vypíše nasledujúci program?

a) 0

b) 1

c) 2

d) rôzny výsledok pri rôznom spustení

```
public class NameGame {
  public static void main(String[] args) {
    Map m = new IdentityHashMap();
    m.put("Mickey", "Mouse");
    m.put("Mickey", "Mantle");
    System.out.println(m.size());
  }
}
```

b)

- java.util.IdentityHashMap
- pool Stringov

Constructors

Skompilujú sa nasledovné triedy?

```
class A {
  private A() {
  }
}

class B extends A {
  B() {
  }
}
```

Constructors

Skompilujú sa nasledovné triedy?

```
class A {
  private A() {
  }
}

class B extends A {
  B() {
  }
}
```

Nie.
Konštruktor B() volá super() cižе A(), ale ten je private.

Constructors

Skompilujú sa nasledovné triedy?

```
class A {
}

class B extends A {
  B() {
  }
}
```

Constructors

Skompilujú sa nasledovné triedy?

```
class A {
}

class B extends A {
  B() {
  }
}
```

Áno.
Konštruktor B() volá super() cize A().
Ten síce nie je uvedený, ale defaultný konštruktor doplní Java sama, pokiaľ v triede iný konštruktor uvedený nie je.

Constructors

Skompilujú sa nasledovné triedy?

```
class A {
  A(int i) {
  }
}

class B extends A {
  B() {
  }
}
```

Constructors

Skompilujú sa nasledovné triedy?

```
class A {
  A(int i) {
  }
}

class B extends A {
  B() {
  }
}
```

Nie.
Konštruktor B() volá super() cižе A(), ale tento Java sama nedoplní keďže v A je už uvedený iný.
Pomôže explicitné doplnenie defaultného konštruktora do A:

```
class A {
  A() {
  }
  A(int i) {
  }
}
```

Constructors

Čo sa vypíše pri volaní new B(15)?

```
class A {
  A() {
    System.out.println("A()");
  }
  A(int i) {
    System.out.println("A(int)");
  }
}

class B extends A {
  B() {
    System.out.println("B()");
  }
  B(int i) {
    System.out.println("B(int)");
  }
}
```

Constructors

Čo sa vypíše pri volaní new B(15)?

```
class A {
  A() {
    System.out.println("A()");
  }
  A(int i) {
    System.out.println("A(int)");
  }
}

class B extends A {
  B() {
    System.out.println("B()");
  }
  B(int i) {
    System.out.println("B(int)");
  }
}
```

A()
B(int)

Constructors

Čo sa vypíše pri volaní new B(15)?

```
class A {
  A() {
    System.out.println("A()");
  }
  A(int i) {
    System.out.println("A(int)");
  }
}

class B extends A {
  B() {
    System.out.println("B()");
  }
  B(int i) {
    super(i);
    System.out.println("B(int)");
  }
}
```

Constructors

Čo sa vypíše pri volaní new B(15)?

```
class A {
  A() {
    System.out.println("A()");
  }
  A(int i) {
    System.out.println("A(int)");
  }
}

class B extends A {
  B() {
    System.out.println("B()");
  }
  B(int i) {
    super(i);
    System.out.println("B(int)");
  }
}
```

A(int)
B(int)

Constructors

Čo sa vypíše pri volaní new B()?

```
class A {
  A() {
    System.out.println("A()");
  }
  A(int i) {
    System.out.println("A(int)");
  }
}

class B extends A {
  B() {
    super(15);
    System.out.println("B()");
  }
  B(int i) {
    System.out.println("B(int)");
  }
}
```

Constructors

Čo sa vypíše pri volaní new B()?

```
class A {
  A() {
    System.out.println("A()");
  }
  A(int i) {
    System.out.println("A(int)");
  }
}

class B extends A {
  B() {
    super(15);
    System.out.println("B()");
  }
  B(int i) {
    System.out.println("B(int)");
  }
}
```

A(int)
B()

Constructors

Čo sa vypíše pri volaní new B()?

```
class A {
  A() {
    System.out.println("A()");
  }
  A(int i) {
    System.out.println("A(int)");
  }
}

class B extends A {
  B() {
    this(15);
    System.out.println("B()");
  }
  B(int i) {
    super(i);
    System.out.println("B(int)");
  }
}
```

Constructors

Čo sa vypíše pri volaní new B()?

```
class A {
  A() {
    System.out.println("A()");
  }
  A(int i) {
    System.out.println("A(int)");
  }
}

class B extends A {
  B() {
    this(15);
    System.out.println("B()");
  }
  B(int i) {
    super(i);
    System.out.println("B(int)");
  }
}
```

A(int)
B(int)
B()

Constructors

More of the same

Čo vypíše nasledujúci program?

a) 0

b) 1

c) 2

d) rôzny výsledok pri rôznom spustení

```
public class Names {
  private Map m = new HashMap();
  public void Names() {
    m.put("Mickey", "Mouse");
    m.put("Mickey", "Mantle");
  }
  public int size() {
    return m.size();
  };
  public static void main(String[] args) {
    Names names = new Names();
    System.out.println(names.size());
  }
}
```

Constructors

More of the same

Čo vypíše nasledujúci program?

a) 0

b) 1

c) 2

d) rôzny výsledok pri rôznom spustení

```
public class Names {
  private Map m = new HashMap();
  public void Names() {
    m.put("Mickey", "Mouse");
    m.put("Mickey", "Mantle");
  }
  public int size() {
    return m.size();
  }
  public static void main(String[] args) {
    Names names = new Names();
    System.out.println(names.size());
  }
}
```

a)

Konštruktor nemá typ.

Constructors

No pain, no gain

Čo vypíše nasledujúci program?

a) Pain, Gain, alebo Main (náhodne)

b) Pain alebo Main (náhodne)

c) Main (vždy)

d) nič z uvedeného

```
public class Rhymes {
  private static Random rnd = new Random();
  public static void main(String[] args) {
    StringBuffer word = null;
    switch (rnd.nextInt(2)) {
      case 1:
        word = new StringBuffer('P');
      case 2:
        word = new StringBuffer('G');
      default:
        word = new StringBuffer('M');
    }
    word.append('a');
    word.append('i');
    word.append('n');
    System.out.println(word);
  }
}
```

Constructors

No pain, no gain

Čo vypíše nasledujúci program?

a) Pain, Gain, alebo Main (náhodne)

b) Pain alebo Main (náhodne)

c) Main (vždy)

d) nič z uvedeného

```
public class Rhymes {
  private static Random rnd = new Random();
  public static void main(String[] args) {
    StringBuffer word = null;
    switch (rnd.nextInt(2)) {
      case 1:
        word = new StringBuffer('P');
      case 2:
        word = new StringBuffer('G');
      default:
        word = new StringBuffer('M');
    }
    word.append('a');
    word.append('i');
    word.append('n');
    System.out.println(word);
  }
}
```

d)

Vypíše sa len 'ain', pretože sa použije konštruktor:

StringBuffer

public **StringBuffer**(int length)

Constructs a string buffer with no characters in it and an initial capacity specified by the length argument.

Exceptions

Majme triedy:

```
public class MyException extends Exception {
  public MyException(String s) {
    this.msg = s;
  }
  public String msg;
}

public class C {
  public void m1(int a) {
    if (a > 3)
      throw new MyException("a > 3");
    System.out.println("A");
  }
}
```

Ako musíme upraviť triedu C aby bola skompilovateľná?

Exceptions

Majme triedy:

```
public class MyException extends Exception {
  public MyException(String s) {
    this.msg = s;
  }
  public String msg;
}

public class C {
  public void m1(int a) throws MyException {
    if (a > 3)
      throw new MyException("a > 3");
    System.out.println("A");
  }
}
```

Ako musíme upraviť triedu C aby bola skompilovateľná?

Exceptions

Majme tie isté triedy:

```
public class MyException extends Exception {
  public MyException(String s) {
    this.msg = s;
  }
  public String msg;
}

public class C {
  public void m1(int a) {
    if (a > 3)
      throw new MyException("a > 3");
    System.out.println("A");
  }
}
```

Je možné nejako zmeniť triedu MyException aby bola trieda C skompilovateľná?

Exceptions

Majme tie isté triedy:

```
public class MyException extends RuntimeException {
  public MyException(String s) {
    this.msg = s;
  }
  public String msg;
}

public class C {
  public void m1(int a) {
    if (a > 3)
      throw new MyException("a > 3");
    System.out.println("A");
  }
}
```

Je možné nejako zmeniť triedu MyException aby bola trieda C skompilovateľná?

Exceptions

Je trieda C skompilovateľná?

```
public class C {
  public void m2(int a) {
    try {
      if (a > 3)
        throw new MyException("a > 3");
    } catch (Exception e) {
      System.out.println("E");
    } catch (MyException e) {
      System.out.println("ME");
    }
  }
}
```

Exceptions

Je trieda C skompilovateľná?

```
public class C {
  public void m2(int a) {
    try {
      if (a > 3)
        throw new MyException("a > 3");
    } catch (Exception e) {
      System.out.println("E");
    } catch (MyException e) {
      System.out.println("ME");
    }
  }
}
```

Nie.
Unreachable catch block for MyException. It is already handled by the catch block for Exception.
Ak chceme MyException špeciálne ošetriť, musíme prehodiť poradie ich odchytávania:

```
} catch (MyException e) {
   System.out.println("ME");
} catch (Exception e) {
   System.out.println("E");
}
```

Exceptions

Ktorá MyException sa vyhodí z metódy m3 pri jej volaní?

```
public class C {
  public void m3(int a) throws MyException {
    try {
      throw new MyException("1");
    } catch (Exception e) {
      throw new MyException("2");
    }
  }
}
```

Exceptions

Ktorá MyException sa vyhodí z metódy m3 pri jej volaní?

```
public class C {
  public void m3(int a) throws MyException {
    try {
      throw new MyException("1");
    } catch (Exception e) {
      throw new MyException("2");
    }
  }
}
```

```
new MyException("2")
```

Exceptions

Ktorá MyException sa vyhodí z metódy m4 pri jej volaní?

```
public class C {
  public void m4(int a) throws MyException {
    try {
      throw new MyException("1");
    } catch (Exception e) {
      throw new MyException("2");
    } finally {
      throw new MyException("3");
    }
  }
}
```

Exceptions

Ktorá MyException sa vyhodí z metódy m4 pri jej volaní?

```
public class C {
  public void m4(int a) throws MyException {
    try {
      throw new MyException("1");
    } catch (Exception e) {
      throw new MyException("2");
    } finally {
      throw new MyException("3");
    }
  }
}
```

```
new MyException("3")
```

Nested classes

Nested class

```
class EnclosingClass {
  ...
  class ANestedClass {
    ...
  }
}
```

A nested class is a class that is a member of another class.

- sytakticky je kód nested class v rámci inej (obaľujúcej) triedy
- úzko súvisí s obaľujúcou triedou resp. ani nemá bez nej zmysel
- má prístup aj k private členom obaľujúcej triedy

Nested classes

Inner class

```
class EnclosingClass {
  ...
  class InnerClass {
    ...
  }
}
```

An inner class is a nested class whose instance exists within an instance of its enclosing class and has direct access to the instance members of its enclosing instance.

- sémantický vzťah
- inštancia môže existovať len v rámci inštancie obaľujúcej triedy
- má prístup k inštančným členom obaľujúcej triedy
- nemôže obsahovať statické členy

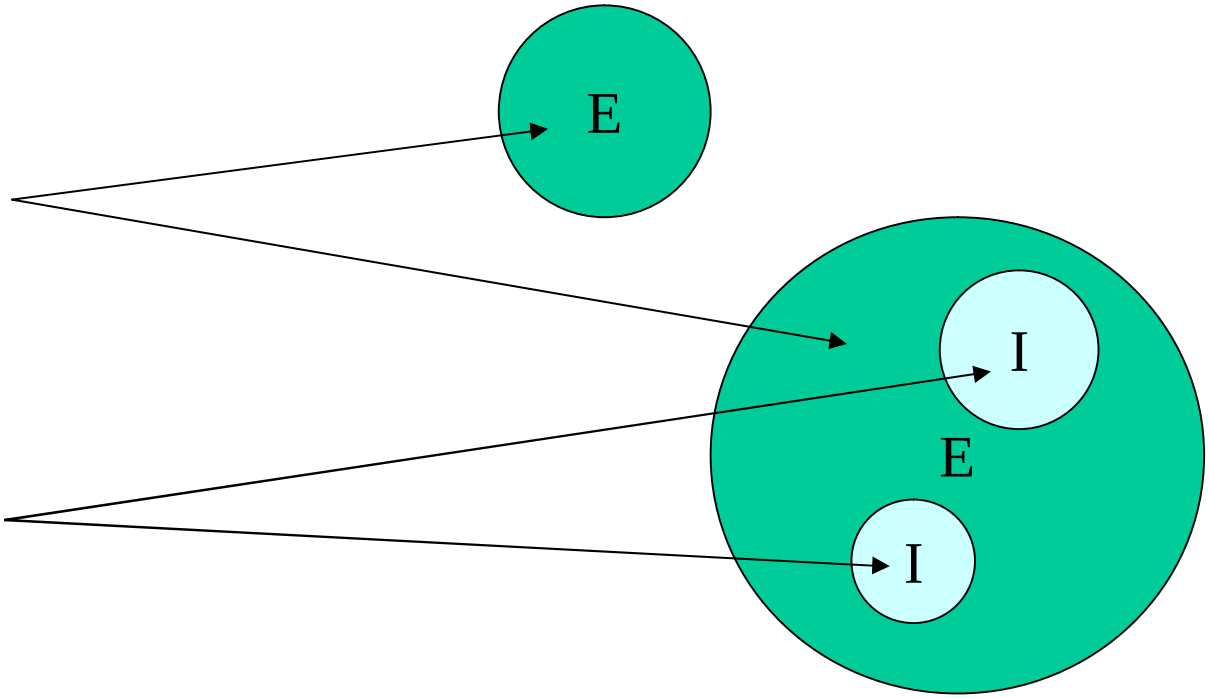
Nested classes

Inner class

```
class EnclosingClass {
  ...
  class InnerClass {
    ...
  }
}
```

Inštancie EnclosingClass sa vyskytujú voľne v pamäti.

Inštancie InnerClass sa môžu vyskytovať len v rámci inštancie EnclosingClass.



Nested classes

Inner class

Skompilujú sa triedy?

```
class Outer {
  class Inner {
  }
}

class Another {
  public void amethod() {
    Inner i = new Inner();
  }
}
```

Nested classes

Inner class

Skompilujú sa triedy?

```
class Outer {
  class Inner {
  }
}

class Another {
  public void amethod() {
    Inner i = new Inner();
  }
}
```

Nie.
Inštancia Inner môže existovať len v rámci inštancie Outer.

Nested classes

Inner class

Skompilujú sa triedy?

```
class Outer {
  public void mymethod() {
    Inner i = new Inner();
  }
  public class Inner {
  }
}
```

Nested classes

Inner class

Skompilujú sa triedy?

```
class Outer {
  public void mymethod() {
    Inner i = new Inner();
  }
  public class Inner {
  }
}
```

Áno.

Nested classes

Inner class

Skompilujú sa triedy?

```
class Outer {
  public class Inner {
  }
}

class Another {
  public void amethod() {
    Outer.Inner i = new Outer().new Inner();
  }
}
```

Nested classes

Inner class

Skompilujú sa triedy?

```
class Outer {
  public class Inner {
  }
}

class Another {
  public void amethod() {
    Outer.Inner i = new Outer().new Inner();
  }
}
```

Áno.

Nested classes

Inner class

Skompiluje sa trieda?

```
class Outer {
  public class Inner {
    static int i = 0;
  }
}
```

Nested classes

Inner class

Skompiluje sa trieda?

```
class Outer {
  public class Inner {
    static int i = 0;
  }
}
```

Nie.
Inner class nemôže obsahovať statické členy.

Nested classes

Inner class

Aký je výstup programu?

```
class Outer {
  private int i = 1;
  class Inner {
    private int i = 2;
    public void mymethod() {
      System.out.println(i);
      System.out.println(Outer.this.i);
    }
  }
}

public class MyTest {
  public static void main(String[] args) {
    new Outer().new Inner().mymethod();
  }
}
```

Nested classes

Inner class

Aký je výstup programu?

```
class Outer {
  private int i = 1;
  class Inner {
    private int i = 2;
    public void mymethod() {
      System.out.println(i);
      System.out.println(Outer.this.i);
    }
  }
}

public class MyTest {
  public static void main(String[] args) {
    new Outer().new Inner().mymethod();
  }
}
```

2

1

Nested classes

Static nested class

```
class EnclosingClass {
  ...
  static class StaticNestedClass {
    ...
  }
  class InnerClass {
    ...
  }
}
```

A static nested class is a nested class whose instance can exists without an instance of its enclosing class.

- inštancia môže existovať aj bez inštancie obaľujúcej triedy
- má prístup k statickým členom obaľujúcej triedy
- prístup k inštančným členom obaľujúcej triedy má len cez jej inštanciu

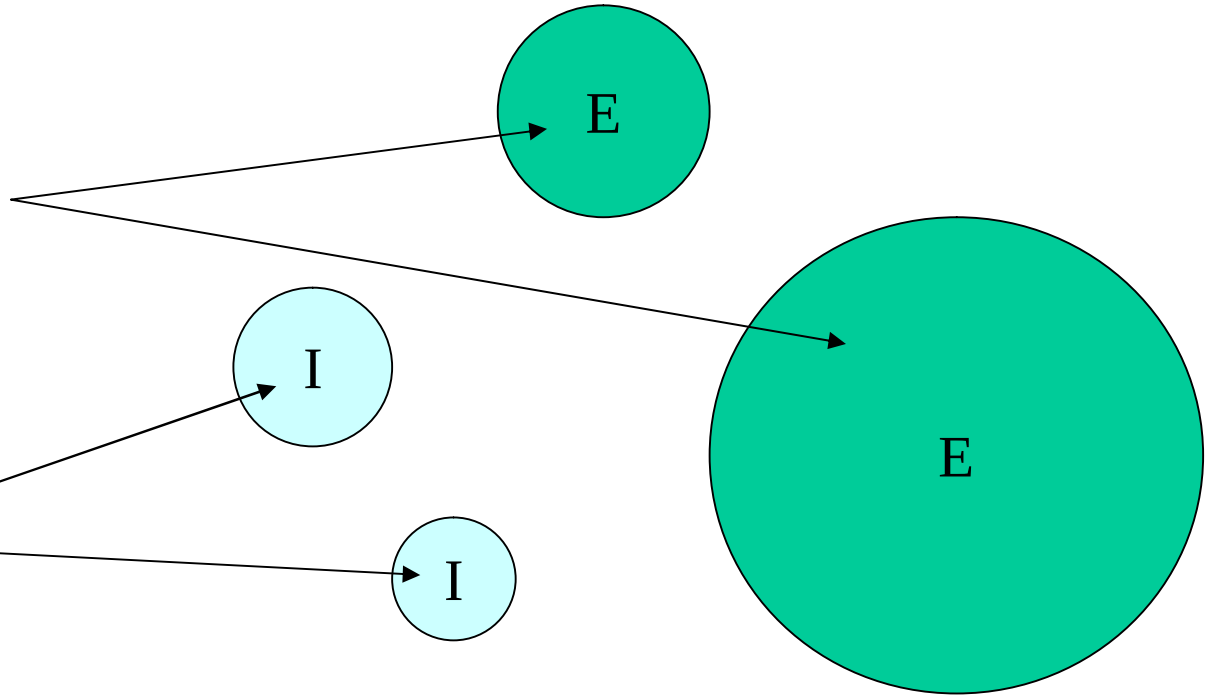
Nested classes

Static nested class

```
class EnclosingClass {
  ...
  static class StaticNestedClass {
    ...
  }
}
```

Inštancie EnclosingClass sa vyskytujú voľne v pamäti.

Inštancie StaticNestedClass sa vyskytujú voľne v pamäti.



Nested classes

Static nested class

Skompilujú sa triedy?

```
class Nest {
  static class NestIn {
  }
}

class Another {
  void amethod() {
    new Nest.NestIn();
  }
}
```

Nested classes

Static nested class

Skompilujú sa triedy?

```
class Nest {
  static class NestIn {
  }
}

class Another {
  void amethod() {
    new Nest.NestIn();
  }
}
```

Áno.

Nested classes

Static nested class

Skompiluje sa táto trieda?

```
class Nest {

  static class NestIn {
    private void test(Nest nest) {
      nest.test();
      System.out.println(nest.i);
    }
  }

  private int i = 1;

  private void test() {
    System.out.println(i);
  }
}
```

Nested classes

Static nested class

Skompiluje sa táto trieda?

```
class Nest {

  static class NestIn {
    private void test(Nest nest) {
      nest.test();
      System.out.println(nest.i);
    }
  }

  private int i = 1;

  private void test() {
    System.out.println(i);
  }
}
```

Áno.

Nested classes

Static nested class

Nastane na niektorom riadku kompilačná chyba?

```
1  public class Nest {
2
3    static class NestIn {
4      private void test(Nest nest) {
5        nest.test();
6      }
7    }
8
9    private void test() {
10     new NestIn().test(this);
11   }
12
13   public static void main(String[] args) {
14     new Nest.NestIn().test(new Nest());
15   }
16 }
```

Nested classes

Static nested class

Nastane na niektorom riadku kompilačná chyba?

```
1  public class Nest {
2
3    static class NestIn {
4      private void test(Nest nest) {
5        nest.test();
6      }
7    }
8
9    private void test() {
10     new NestIn().test(this);
11   }
12
13   public static void main(String[] args) {
14     new Nest.NestIn().test(new Nest());
15   }
16 }
```

Program je skompilovateľný.
Skončí však chybou
java.lang.StackOverflowError.

Nested classes

Anonymous inner class

An anonymous inner class is an inner class without name.

- používame len pre malé jednoduché triedy, inak robia kód neprehľadným
- časté použitie ako implementácia interfaceov v AWT a pod.

Nested classes

Anonymous inner class

- inner class with name X

```
class EnclosingClass {
  class X implements Iface {
    public void method() {
      ...
    }
  }
  Iface cmp = new X();
}
```

- anonymous inner class

```
class EnclosingClass {
  Iface cmp = new Iface() {
    public void method() {
      ...
    }
  };
}
```

Nested classes

Anonymous inner class

- použitie ako parameter metódy

```
button.addMouseListener(new java.awt.event.MouseAdapter() {
  public void mouseClicked(java.awt.event.MouseEvent e) {
    ...
  }
});
```

```
java.util.Arrays.sort(arr, new java.util.Comparator() {
  public int compare(Object o1, Object o2) {
    int i1 = ((Integer) o1).intValue();
    int i2 = ((Integer) o2).intValue();
    return i1 < i2 ? -1 : (i1 == i2 ? 0 : 1);
  }
});
```

Nested classes

Nested class in method

- nested class môže byť definovaná aj v rámci nejakej metódy
- má prístup len k final premenným a final parametrom tejto metódy

Nested classes

Nested class in method

```
public class Problem {

  public static void main(String[] args) {
    final Timer timer = new Timer();
    timer.schedule(new TimerTask() {
      public void run() {
        System.out.println("Exiting.");
        timer.cancel();
      }
    }, 5000);

    System.out.println("In 5 seconds this application will exit.");
  }
}
```

- anonymous inner class definovaná v rámci metódy main vidí premennú timer metódy main vďaka tomu že je označená ako final

- bez slovíčka final by bol kód neskompilovateľný

Nested classes

Visibility

Ku ktorým premenným je možné pristupovať v metóde iMethod(int)?

```
public class Outer {
  public int a = 1;
  private int b = 2;
  public void method(final int c) {
    int d = 3;
    class Inner {
      private void iMethod(int e) {
      }
    }
  }
}
```

Nested classes

Visibility

Ku ktorým premenným je možné pristupovať v metóde iMethod(int)?

```
public class Outer {
  public int a = 1;
  private int b = 2;
  public void method(final int c) {
    int d = 3;
    class Inner {
      private void iMethod(int e) {
      }
    }
  }
}
```

a

b

c

e

Ďakujem za pozornosť