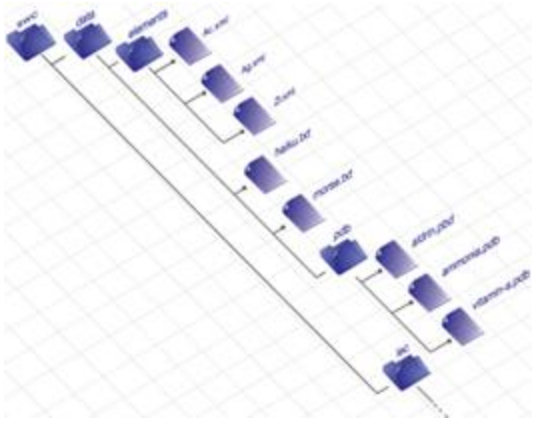
# 7. prednáška (8.11.2021)

```
Exception in thread "main" java.lang.NullPointerException
        at Vynimkarka.kladnyPriemer(Vynimkarka.java:9)
        at Spustac.main(Spustac.java:10)
```

# Výnimky I,

# adresáre a súbory



## alebo

## Pomaly opúšťame

## korytnačky


# Ľahký štart

- Metód prefixSum, ktorá vráti súčet prvých k prvkov v poli.

```
public int prefixSum(int[] numbers, int k) {
      int result = 0;
      for (int i = 0; i < k; i++) {
            result += numbers[i];
      }
      return result;
}
```

Vyskúšajme rôzne (zákerné?) vstupy


# Červené hlášky...

- Výpis v prípade chyby

Názov ???

```
Exception in thread "main“
java.lang.ArrayIndexOutOfBoundsException: 5
at SmartTurtle.prefixSum(SmartTurtle.java:10)
at Launcher.main(Launcher.java:11)
```

**Stack trace** = „miesto pri behu programu“

`prefixSum` bol volaný z metódy `main` v triede `Launcher` z 11. riadku

Program skončil vykonávanie na 10. riadku v triede `SmartTurtle` a bolo to v metóde `prefixSum`


# ArrayIndexOutOfBoundsException

## Class ArrayIndexOutOfBoundsException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.IndexOutOfBoundsException
          java.lang.ArrayIndexOutOfBoundsException

**All Implemented Interfaces:**

Serializable

---

```
public class ArrayIndexOutOfBoundsException
extends IndexOutOfBoundsException
```

Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.


# Čo sú to výnimky?

- **Výnimky**
  - **špeciálne objekty**
  - vznikajú vo **výnimočných stavoch**, keď nejaké metódy nemôžu prebehnúť štandardným spôsobom alebo nevedia vrátiť očakávanú hodnotu
  - takmer všetky moderné programovacie jazyky signalizujú výnimočný (neočakávaný) stav vo forme výnimiek

Experiment: Ako sa prejaví výnimka okrem „červeného výpisu“?

# Keď sa hodí výnimka...

Dopad na bežiace metódy

Výnimočný stav **hádže výnimku** (inštanciu vhodnej výnimkovej triedy)

Ak sa výnimka objaví v metóde, metóda je okamžite ukončená.


# java.lang.NullPointerException

- Najčastejšia výnimka?
- Príklady:

```
private Turtle t;

t.step(100);
```

`null.step(100)`

---

```
private int[] pole;

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

`null.length`

---

```
Turtle[] korytnacky = new Turtle[10];

korytnacky[0].turn(90);
```

`null.turn(90)`


# Ďalšie výnimky

- `java.lang.ArithmeticException: / by zero`
  - delenie celého čísla nulou (pozor, pri `double`!)
- `java.lang.NegativeArraySizeException`
  - `int[] pole = new int[-5];`
- `java.lang.StringIndexOutOfBoundsException`
  - prístup ku znaku na neexistujúcom indexe

**Prevencia:**

Všetky tieto výnimky sa dajú ošetriť `if`-mi.

# V dokumentácii

- možné výnimky sú uvedené pod **Throws:**

### charAt

```
public char charAt(int index)
```

Returns the `char` value at the specified index. An index ranges from `0` to `length()` index `0`, the next at index `1`, and so on, as for array indexing.

If the `char` value specified by the index is a surrogate, the surrogate value is returne

**Specified by:**

`charAt` in interface `CharSequence`

**Parameters:**

`index` - the index of the `char` value.

**Returns:**

the `char` value at the specified index of this string. The first `char` value is at inde

**Throws:**

`IndexOutOfBoundsException` - if the `index` argument is negative or not less


Nemusia byť uvedené všetky možné výnimky


# Maximum zo Stringu

- Chceme nájsť **najväčšie číslo** v reťazci medzerami oddelených čísel

```
"125 26 1587 11 0 15"
```

- Potrebujeme :
  - Rozdeliť vstup na „slová“
  - Každé „slovo“ parsovať na číslo

```
int number = Integer.parseInt("...");
```


# Maximum zo Stringu

- Chceme nájsť **najväčšie číslo** v reťazci medzerami oddelených čísel

```
"125 26 1587 11 0 15"
```

- A čo ak v reťazci nie sú len čísla?

```
"125 Java 1587 1a1 0 15"
```


# Maximum zo Stringu

- Čo sa stane, keď:

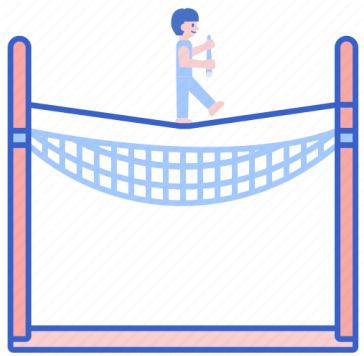
```
number = Integer.parseInt("Java");

number = Integer.parseInt("");
```

- Vyhodená výnimka **NumberFormatException**
- Prevencia? If?

# Odchytávame výnimky

- Výnimky vieme **odchytiť**


```
try {
  // blok príkazov,
  // kde môžu vzniknúť výnimky,
  // ktoré si trúfame odchytiť
} catch (TypVýnimky1 e) {
  // vysporiadanie sa s daným
  typom výnimky
} catch (TypVýnimky2 e) {
  // vysporiadanie sa s daným typom výnimky
}
```




# Situácia 1

```
príkaz1;
try {
  príkaz2;
  príkaz3;
} catch (TypVýnimky1 e) {
  príkazE1;
} catch (TypVýnimky2 e) {
  príkazE2;
}
príkaz4;
```

V prípade normálneho priebehu sa vykonajú:

```
príkaz1;
príkaz2;
príkaz3;
príkaz4;
```


# Situácia 2

```
príkaz1;
try {
  príkaz2;
  príkaz3;
} catch (TypVýnimky1 e) {
  príkazE1;
} catch (TypVýnimky2 e) {
  príkazE2;
}
príkaz4;
```

Nech je výnimka akákoľvek, končíme a nič viac sa z metódy nevykoná (výnimka sa hádže ďalej do metódy, ktorá túto metódu volala)

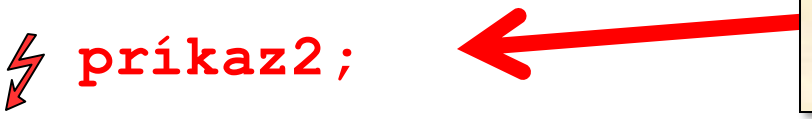

# Situácia 3

```
príkaz1;
try {
  príkaz2;
  príkaz3;
} catch (TypVýnimky1 e) {
  príkazE1;
} catch (TypVýnimky2 e) {
  príkazE2;
}
príkaz4;
```

Ak sa vyhodí výnimka triedy `TypVýnimky1` vykonáme `príkazE1` a `príkaz4`


# Situácia 4

```
príkaz1;
try {
  príkaz2;
  príkaz3;
} catch (TypVýnimky1 e) {
  príkazE1;
} catch (TypVýnimky2 e) {
  príkazE2;
}
príkaz4;
```



Ak sa vyhodí výnimka triedy `TypVýnimky2` vykonáme `príkazE2` a `príkaz4`

# Situácia 5

```
príkaz1;
try {
  príkaz2;
  príkaz3;
} catch (TypVýnimky1 e) {
  príkazE1;
} catch (TypVýnimky2 e) {
  príkazE2;
}
príkaz4;
```

Ak sa vyhodí výnimka inej triedy ako `TypVýnimky1` alebo `TypVýnimky2` končíme a výnimka sa hádže ďalej


# Blok finally

- **finally**
  - príkazy, ktoré sa vykonajú **vždy**, ak už program vošiel do bloku **try** - bez ohľadu na to, či v bloku **try** výnimka nastala alebo nenastala, bez ohľadu na to, či sme ju odchytili alebo nie.

```
try {
  // ...
} catch (TypVýnimky1 e) {
  // ...
} catch (TypVýnimky2 e) {
  // ...
} finally {
  // príkazy, ktoré sa vykonajú bez ohľadu na to, čo sa stalo
}
```

**vždy**

# Prečo finally?

- `finally` sa využíva na „upratanie“ – či už veci dopadli dobre, alebo nie.

- Typické použitie bloku `finally`:
  - Na uzavretie súboru
  - Na ukončenie sieťového pripojenia
  - Na ukončenie pripojenia na databázu
  - Na zápis vykonanej operácie do logovacieho súboru


# Slajd pre fajnšmekrov

- Čo ak nastane výnimka v bloku `catch`, alebo v bloku `finally`?


# Výnimky v Jave

- **Nekontrolované** (unchecked)
  - nemusíme ich odchytávať

Zatiaľ sme stretli len takéto

- **Kontrolované** (checked)
  - musíme ich odchytávať
  - nútia programátora, aby v kóde nezabudol na problémovú situáciu, ktorá reálne môže nastať

Také ešte stretneme

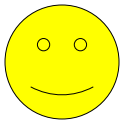
- O type výnimiek rozhoduje ich tvorca.


# Sumarizácia

- Čo sa dá ošetriť `if`-mi, ošetrujeme `if`-mi!
- Ak nastane výnimka mimo **`try`** bloku,
  - výnimka sa šíri ďalej...

- Ak nastane výnimka v **`try`** bloku a neodchytíme ju v **`catch`** bloku,
  - program skočí do **`finally`** bloku, ten sa vykoná
  - výnimka sa šíri ďalej...

- Ak nastane výnimka v **`try`** bloku a odchytíme ju v **`catch`** bloku,
  - program skočí do príslušného **`catch`** bloku,
  - potom do **`finally`** bloku
  - a potom pokračujeme v programe ďalej



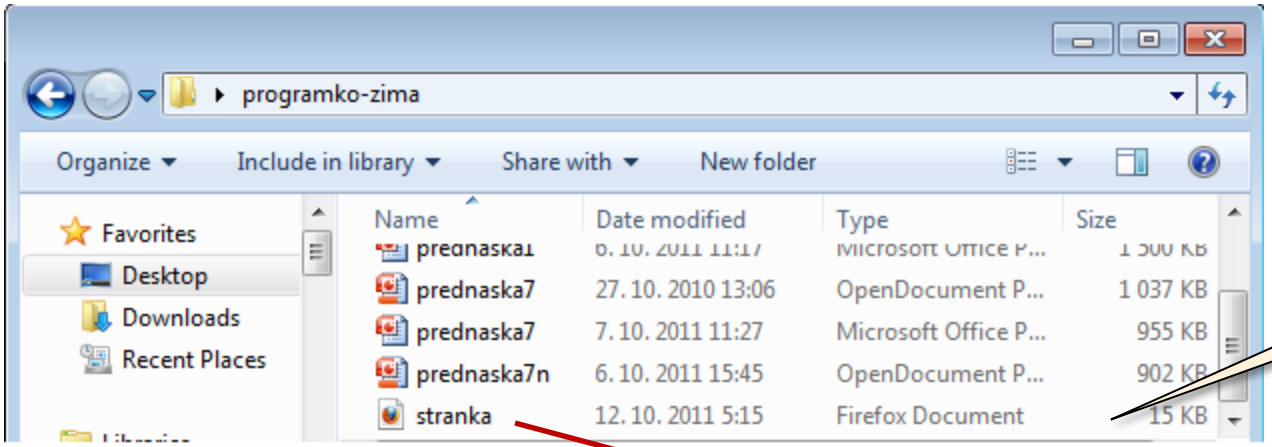

# Vstupno-výstupné operácie

- Programy potrebujú komunikovať s okolím
  - získavať z neho údaje
  - odovzdávať/zobrazovať mu údaje
- Potrebujú vstupy a výstupy
  - vstup: klávesnica, súbor, myš, internet, databáza, ...
  - výstup: monitor, súbor, internet, tlačiareň, databáza, ...

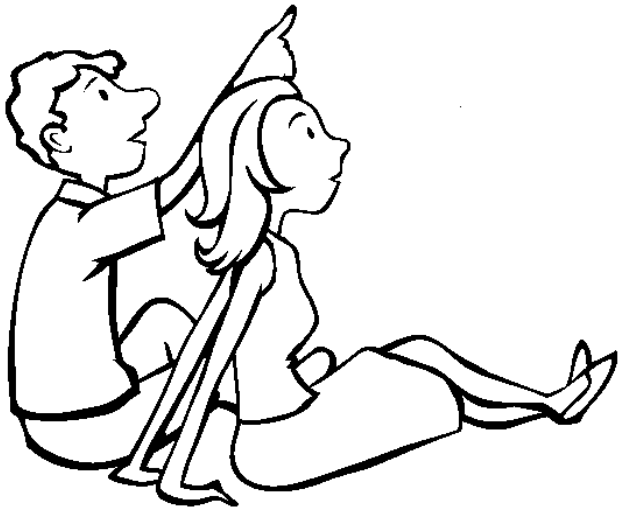

# Adresáre a súbory

- Adresáre a súbory sú dôverne známe z operačných systémov
- V Jave (aj Linuxe)
  - súbor a adresár splývajú do jedného pojmu
  - adresár je tiež súbor
- Súbor to je:
  - Dáta (pohľad z NotePadu): postupnosť čísel tvoriaca obsah súboru
  - Metadáta (pohľad z Total Commandera): názov, veľkosť, umiestnenie, oprávnenia, vlastník, …

# Pohľad na súbor




metadáta



dáta

```
<?xml
version="1.0"
encoding="utf-8"?>
<!DOCTYPE html
PUBLIC "-
//W3C//DTD
XHTML 1.0
Transitional//EN"
...
```


# Adresáre a súbory

- Adresáre tvoria stromovú hierarchiu
- Vo Windowse:
  - Úplný názov súboru: C:\Users\franklin\Documents\rozvrh.png
  - Cesta k súboru: C:\Users\franklin\Documents
  - Názov súboru: rozvrh.png
- V Linuxe
  - Úplný názov súboru: /home/franklin/rozvrh.png
  - Cesta k súboru: /home/franklin
  - Názov súboru: rozvrh.png

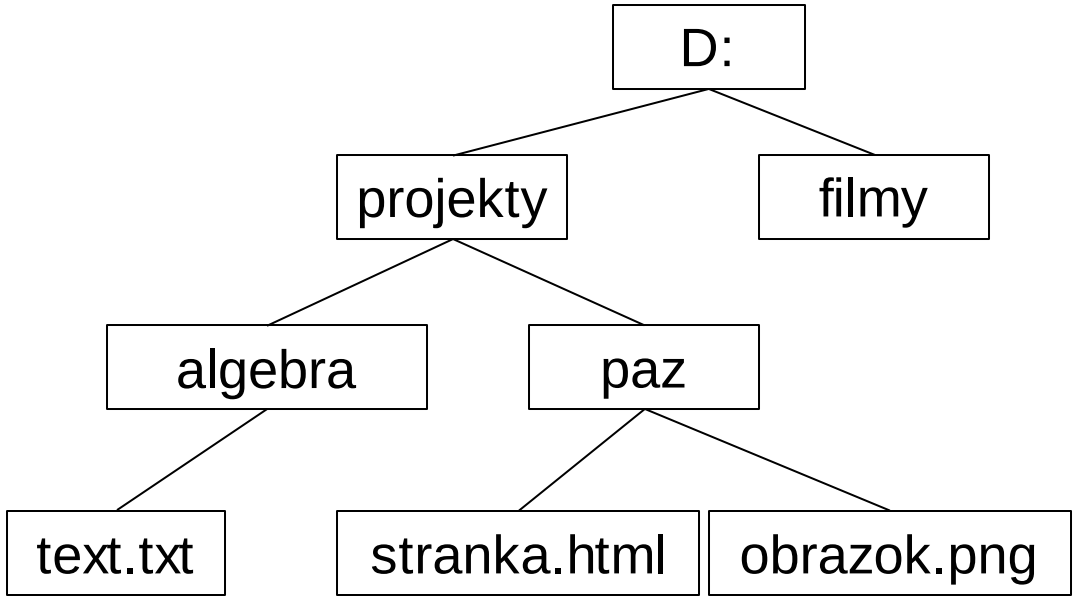

# Cesta k súborom

- Vo Windowse:
  - Položky sú oddelené spätnou lomkou \, ale je možné používať aj /, len o tom málokto vie
  - POZOR, častá chyba: ak chcete používať spätné lomky, v reťazcoch ich musíte zdvojiť ( \ je špeciálny znak - \n, \t)

    “C:\\Users\\franklin\\Documents\\nový priečinok”
- V Linuxe
  - Položky sú oddelené lomkou /


# Cesta k súborom

- Absolútna cesta: **D:\projekty\paz\obrazok.png**
- Relatívna cesta: vzhľadom k nejakému adresáru
  - **..\paz\obrazok.png** je relatívna vzhľadom k **D:\projekty\algebra**




# Aktuálny adresár

- Relatívna cesta môže byť aj k **aktuálnemu adresáru**
- Aktuálny adresár:
  - ak spúšťame program z Eclipse, je to adresár projektu
  - ak spúšťame program z príkazového riadku, je to adresár, z ktorého spúšťame program
- Pre fajnšmekrov:
  - Aktuálny adresár sa dá získať cez:

```
System.getProperty("user.dir");
```

# Trieda `File`

- Objekty triedy `File` uchovávajú cestou k súboru alebo adresáru
  - plus kopa metód na prácu s metadátami o súboroch alebo adresároch
- Tento súbor alebo adresár nemusí reálne existovať !


# Trieda File

```
// úplná cesta k adresáru s použitím spätných lomiek
File adresar = new File("C:\\Windows\\System32");

// úplná cesta k súboru s použitím obyčajných lomiek
File subor1 = new File("C:/Windows/system.ini");

// relatívna cesta k súboru C:\Windows\System32\shell32.dll
   vzhľadom k adresáru C:\Windows\System32
File subor2 = new File(adresar, "shell32.dll");

// relatívna cesta k súboru vzhľadom k aktuálnemu adresáru
File subor3 = new File("heslo.txt");
```


# Niektoré užitočné metódy

| | |
|---|---|
| `String getPath()` | Vráti úplný názov súboru |
| `String getName()` | Vráti názov súboru alebo adresára (bez cesty) |
| `boolean exists()` | Vráti `true`, ak súbor/adresár existuje |
| `boolean isDirectory()` | Zistí, či inštancia zodpovedá adresáru |
| `boolean isFile()` | Zistí, či inštancia zodpovedá súboru |
| `long length()` | Vráti veľkosť súboru |
| `void createNewFile()` | Vytvorí súbor, ak neexistuje. Môže vyvolať výnimku `IOException` ak nemáme dostatočné práva, alebo neexistuje adresár, v ktorom by sa mal tento súbor vytvoriť. |


# Niektoré užitočné metódy

| | |
|---|---|
| `void mkdir()` | Vytvorí adresár, zodpovedajúci poslednej položke v ceste, nadadresár musí existovať. |
| `void mkdirs()` | Vytvorí celú adresárovú štruktúru v ceste. |
| `void renameTo(File)` | Premenuje súbor podľa inej inštancie triedy `File`. |
| `void delete()` | Odstráni súbor. |
| `String[] list()` | Vráti pole názvov súborov/podadresárov v adresári. Vráti `null`, ak nejde o existujúci adresár. |
| `File[] listFiles();` | Vráti pole inštancií triedy `File` zodpovedajúcich súborom/podadresárom v adresári. Vráti `null` ak nejde o existujúci adresár. |

# Testujeme File

- Cieľ:
  - Metóda, ktorá vypíše všetky súbory a adresáre v danom adresári.
  - Metódu, ktorá vypíše mená a veľkosti všetkých mp3 súborov v danom adresári


# Práca s textovými súbormi

- Samotná trieda `File` nám neumožňuje pracovať s dátami v súbore
  - … ale existujú iné triedy.

- Práca s obsahom súborov sa vždy skladá z 3 krokov:
  - **otvorenie** súboru, ktoré sa udeje pri vzniku nejakého čítača alebo zapisovača
  - **práca s obsahom** súboru (teda čítanie alebo zapisovanie)
  - **zatvorenie** súboru

Čo sme otvorili, musíme **VŽDY** zatvoriť.


# Zápis do textového súboru

- Kto?
  - objekty triedy `PrintWriter`
- Ako ho vytvoriť?
  - `new PrintWriter(  );`
    - ak súbor neexistuje, vytvorí sa
    - ak súbor existuje, jeho obsah sa zmaže
- Ako písať?
  - cez metódy `println` a `print` - presne ako ich má `System.out`
- Ako zatvoriť súbor otvorený na zapisovanie?
  - metóda `close`

Objekt triedy `File` s cestou k súboru, kam zapisujeme.


# Vytvorenie zapisovača

**PrintWriter**

```
public PrintWriter(File file)
            throws FileNotFoundException
```

Creates a new PrintWriter, without automatic line flushing, with the specified file. This convenience constructor creates the necessary intermediate `OutputStreamWriter`, which will encode characters using the default charset for this instance of the Java virtual machine.

**Parameters:**

`file` - The file to use as the destination of this writer. If the file exists then it will be truncated to zero size; otherwise, a new file will be created. The output will be written to the file and is buffered.

**Throws:**

`FileNotFoundException` - If the given file object does not denote an existing, writable regular file and a new regular file of that name cannot be created, or if some other error occurs while opening or creating the file

`SecurityException` - If a security manager is present and `checkWrite(file.getPath())` denies write access to the file

**Since:**

1.5


# Hľadanie správnej cesty

- **Riešime výnimku** pri vytvorení PrintWriter-a...
- Čo ak súbor **nezatvoríme**?
- **Kde zatvárať**, aby sme zatvorili otvorený súbor?
  - môže počas zapisovania vzniknúť výnimka?


# Schéma práce s PrintWriter-om

```
File subor = new File("C:\\adresare\\subor.txt");
PrintWriter pw = null;
try {
   pw = new PrintWriter(subor);

   // píšeme do pw

} catch (FileNotFoundException e) {
     System.err.println("Súbor " +
         subor.getName() + " som nenašiel");
} finally {
   if (pw != null)
      pw.close();
}
```


# Zápis do textového súboru

- Prácu s textovým súborom budeme vždy realizovať v rámci **`try-catch`** bloku
- Musíme odchytávať možnú výnimku `FileNotFoundException`
  - musíme = kontrolovaná výnimka
  - vyhodí sa, keď adresár v ktorom má nový súbor vzniknúť neexistuje
    - objekty triedy `File` môžu uchovávať aj neexistujúce cesty
  - vyhodí sa, ak sa súbor s danou cestou nepodarí vytvoriť? Pamätáte si DVD-čka?
- V bloku **`finally`** zatvárame súbor!


# try-catch so zdrojmi

```
try (PrintWriter pw = new PrintWriter(new File(fn))) {

  // práca s PrintWriter-om

} catch (FileNotFoundException e) {
  System.err.println("Nepodarilo sa otvoriť súbor.");
}
```

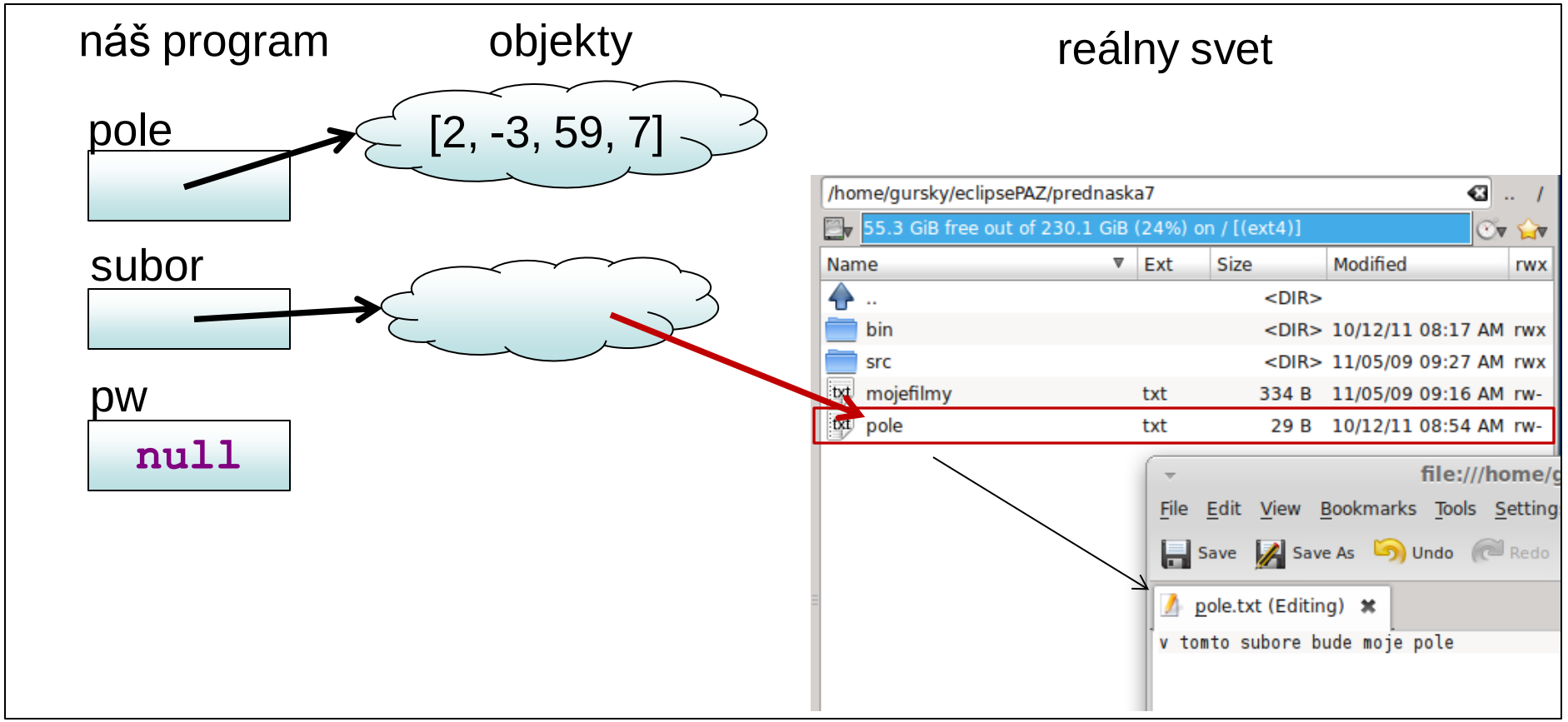
Zatvorenie sa zrealizuje automaticky


# Tréning

- Vytvorme si metódu, ktorá vypíše do zadaného súboru v prvom riadku veľkosť poľa čísiel a v druhom riadku obsah poľa čísiel.

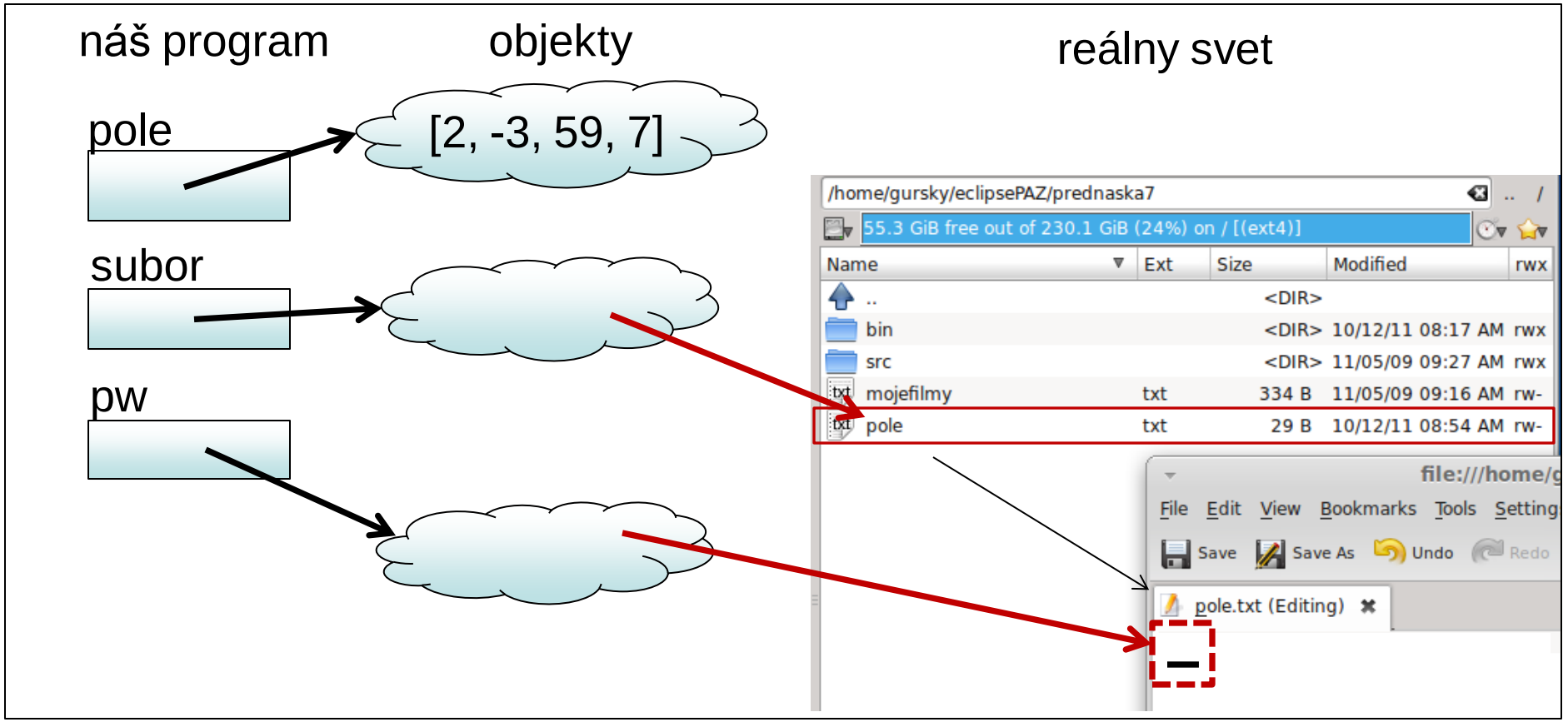
# Zapisujeme do súboru

```
public void saveNumbersToFile(File subor, int[] pole) {
  PrintWriter pw = null;
  try {
    pw = new PrintWriter(subor);
```



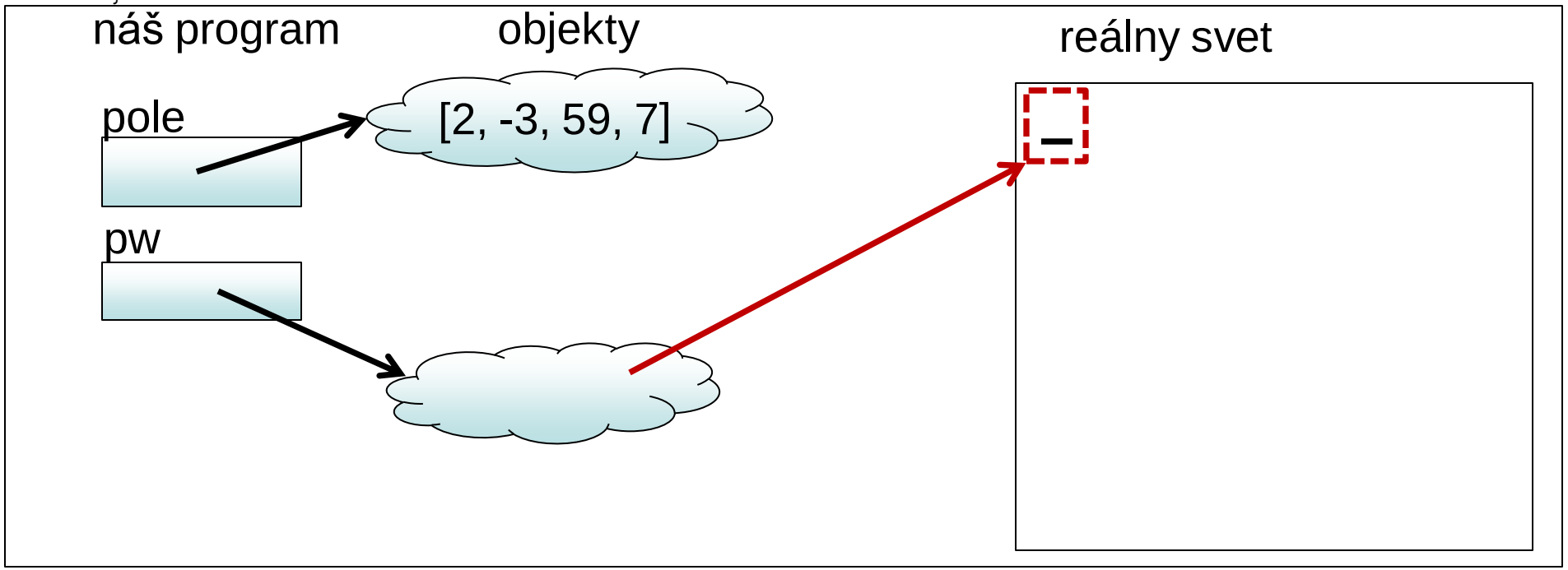
# Zapisujeme do súboru

```
public void saveNumbersToFile(File subor, int[] pole) {
  PrintWriter pw = null;
  try {
    pw = new PrintWriter(subor);
```



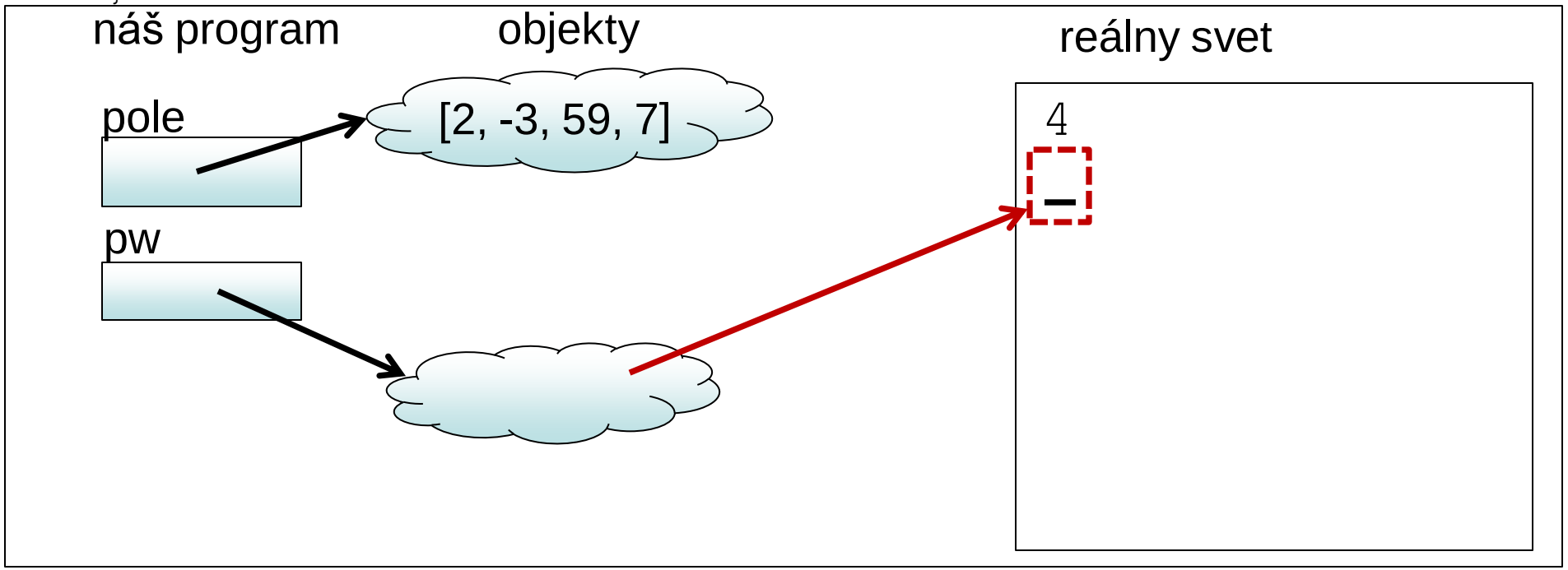
# Zapisujeme do súboru

```
public void saveNumbersToFile(File subor, int[] pole) {
  PrintWriter pw = null;
  try {
    pw = new PrintWriter(subor);
    pw.println(pole.length);
    for (int i = 0; i < pole.length; i++) {
      pw.print(pole[i]+" ");
    }
```



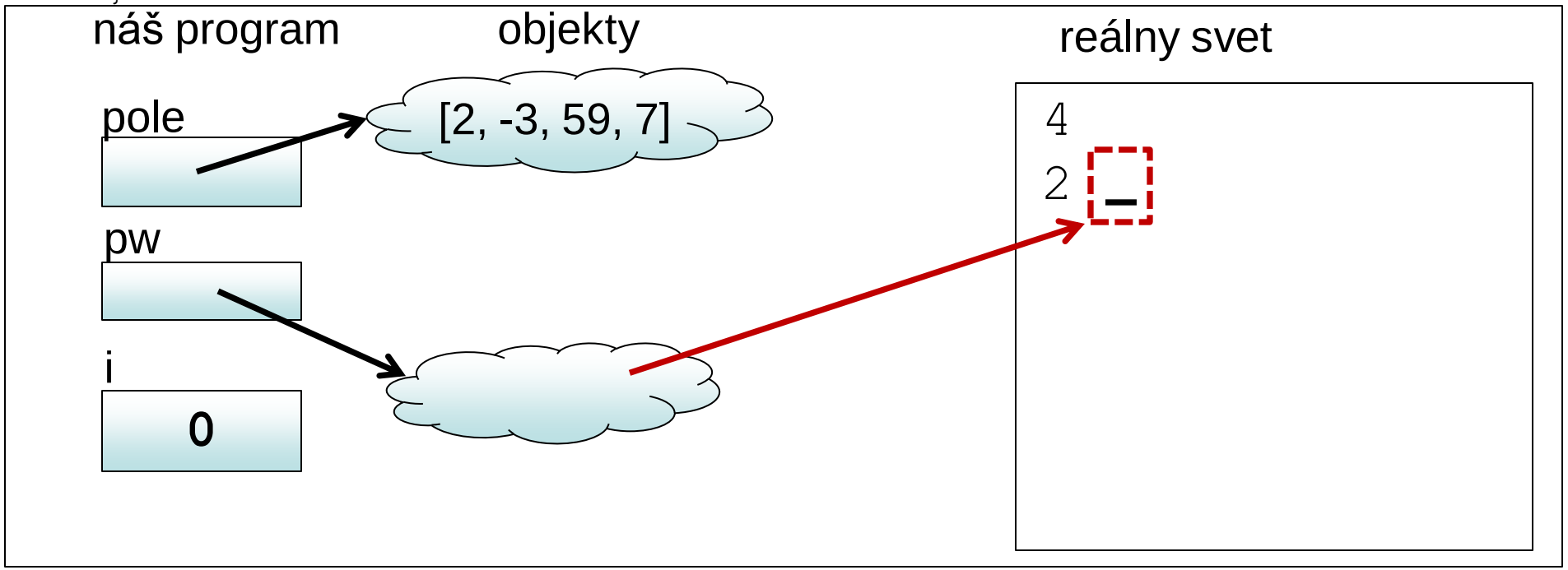
# Zapisujeme do súboru

```
public void saveNumbersToFile(File subor, int[] pole) {
  PrintWriter pw = null;
  try {
    pw = new PrintWriter(subor);
    pw.println(pole.length);
    for (int i = 0; i < pole.length; i++) {
      pw.print(pole[i]+" ");
    }
```



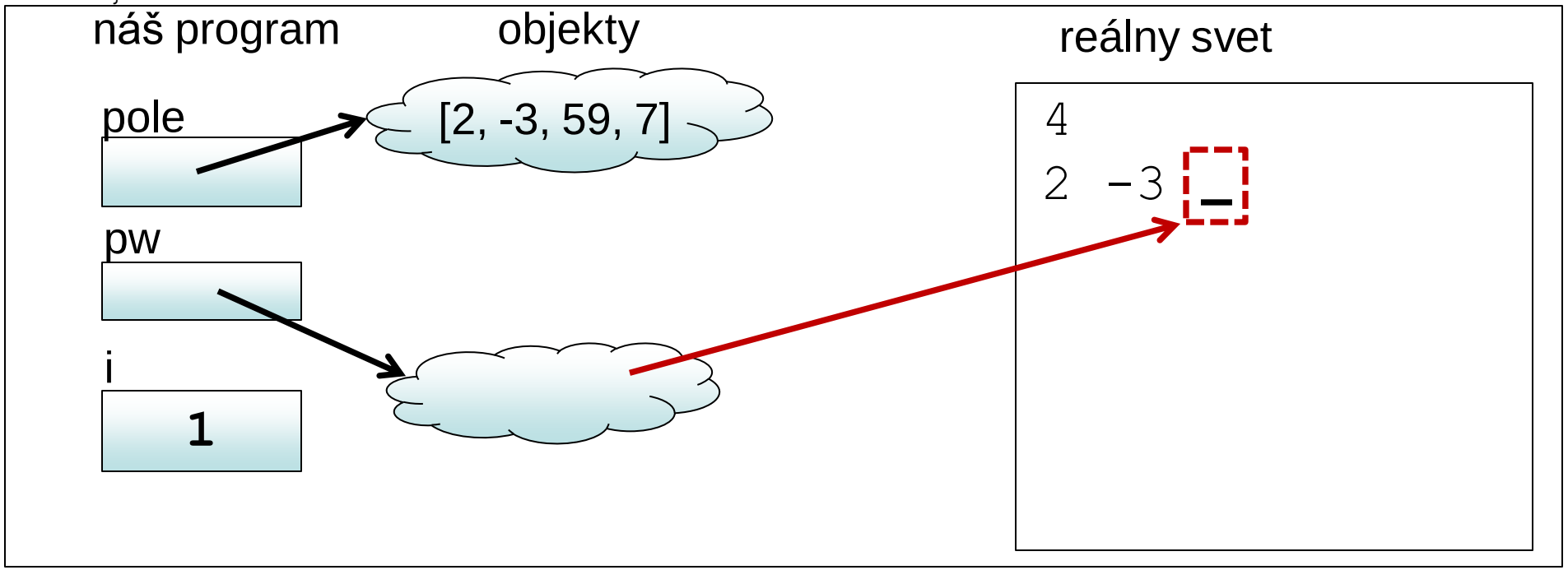
# Zapisujeme do súboru

```
public void saveNumbersToFile(File subor, int[] pole) {
  PrintWriter pw = null;
  try {
    pw = new PrintWriter(subor);
    pw.println(pole.length);
    for (int i = 0; i < pole.length; i++) {
      pw.print(pole[i]+" ");
    }
```



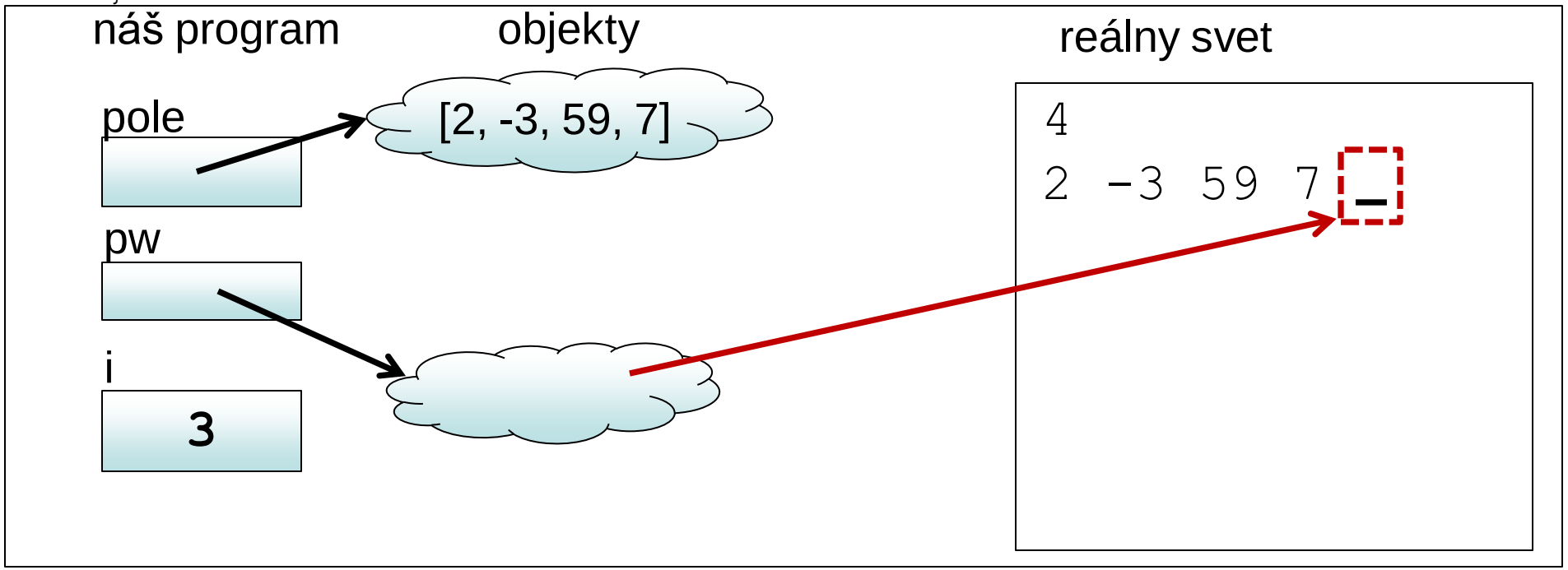
# Zapisujeme do súboru

```
public void saveNumbersToFile(File subor, int[] pole) {
  PrintWriter pw = null;
  try {
    pw = new PrintWriter(subor);
    pw.println(pole.length);
    for (int i = 0; i < pole.length; i++) {
      pw.print(pole[i]+" ");
    }
```



# Zapisujeme do súboru

```
public void saveNumbersToFile(File subor, int[] pole) {
  PrintWriter pw = null;
  try {
    pw = new PrintWriter(subor);
    pw.println(pole.length);
    for (int i = 0; i < pole.length; i++) {
      pw.print(pole[i]+" ");
    }
```




# Zapisujeme do súboru

```
public void saveNumbersToFile(File subor, int[] pole) {
  PrintWriter pw = null;
  try {
    pw = new PrintWriter(subor);
    pw.println(pole.length);
    for (int i = 0; i < pole.length; i++) {
      pw.print(pole[i]+" ");
    }
  } catch (FileNotFoundException e) {
    System.err.println("Súbor " + subor.getName()+ " sa nenašiel");
  } finally {
    if (pw!=null)
      pw.close();
  }
}
```


# Zapisujeme do súboru

```
public void saveNumbersToFile(File subor, int[] pole) {
  try (PrintWriter pw = new PrintWriter(subor)) {
      pw.println(pole.length);
      for (int i = 0; i < pole.length; i++) {
            pw.print(pole[i]+" ");
      }
  } catch (FileNotFoundException e) {
    System.err.println("Súbor " + subor.getName()
            + " sa nenašiel");
  }
}
```


# Ďakujem za pozornosť !