JAVA

java.lang.System

java.lang.System

- obsahuje jen statické elementy
- nelze od ní vytvářet instance

- atributy
  - `java.io.InputStream in`
    - standardní vstup
  - `java.io.PrintStream out`
    - standardní výstup
  - `java.io.PrintStream err`
    - standardní chybový výstup

Metody

- `void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
  - kopíruje pole
  - funguje i pokud src==dest
- `long currentTimeMillis()`
  - aktuální čas v milisekundách od 1.1.1970
  - přesnost záleží na OS
- `long nanoTime()`
  - hodnota systémového timeru v nanosekundách
  - počet nanosekund od *nějakého* pevného času
    - i v budoucnosti, tj. hodnota může být záporná
  - pro počítaní nějaké doby
  - od JDK 5

Metody

- `void exit(int status)`
  - ukončí JVM
- `void gc()`
  - „doporučení“ pro JVM, ať pustí garbage collector
- `void runFinalization()`
  - „doporučení“ pro JVM, ať provede všechny dosud neprovedené a naplánované finalizátory
- `void setIn(InputStream s)`
  `void setOut(PrintStream s)`
  `void setErr(PrintStream s)`
  - znovu nastaví daný vstup nebo výstup
- `int identityHashCode(Object x)`
  - vrátí implicitní hash kód objektu

Properties

- dvojice
  - klíč – hodnota
  - String (klíč i hodnota)
- systémové i vlastní
- `Properties getProperties()`
  - vrátí všechny nastavené properties
  - java.util.Properties – potomek java.util.Hashtable
- `String getProperty(String key)`
  - vrátí hodnotu
  - pokud klíč není nastaven, vrací `null`
- `String getProperty(String key, String def)`
  - vrátí hodnotu
  - pokud klíč není nastaven, vrací `def`

Properties

- `void setProperties(Properties props)`
  - nastaví properties v props
- `String setProperty(String key, String val)`
  - nastaví danou property
  - vrací původní hodnotu nebo null
- `String clearProperty(String key)`
  - zruší danou property

- nastavení properties při startu JVM
  - parametr `-Dkey=value`
  - př. `java -DdefaultDir=/usr Program`
- typicky se jako klíče používají hierarchická jména oddělená tečkami

Vždy nastavené properties

- java.version
- java.home
  - adresář, ve kterém je instalace JAVy
- java.class.path
- java.io.tmpdir
  - kde se budou vytvářet dočasné soubory
- os.name, os.architecture, os.version
  - identifikace operačního systému
- file.separator
  - oddělovač souborů v cestě (unix "/", win "\")
- path.separator
  - oddělaoč cest (unix ":", win ";")
- line.separator
  - oddělovač řádků (unix "LF", win "CR LF")

Vždy nastavené properties

- user.name
  - název učtu aktuálního uživatele
- user.home
  - domovský adresář
- user.dir
  - aktuální adresář
- dále několik properties identifikujích VM

Proměnné prostředí

- **`Map<String, String> getenv()`**
  - všechny proměnné prostředí
  - nemodifikovatelná kolekce
- **`String getenv(String name)`**
  - proměnná s daným jménem nebo null

java.lang.Runtime

Runtime

- vždy existuje jedna instance
  - nelze vytvářet další instance
- `Runtime getRuntime()`
  - statická metoda
  - vrátí instanci Runtime
- `int availableProcessors()`
  - závisí na implementaci
  - vrácená hodnota se může za běhu programu měnit
- `long freeMemory()`
  - volná paměť dostupná pro JVM
- `long maxMemory()`
  - maximální dostupná paměť pro JVM
- `void halt(int status)`
  - ukončení JVM, na nic nečeká

Runtime

- `void addShutdownHook(Thread hook)`
  - nastaví vlákno, které se má provést při ukončování JVM
  - hook – vytvořené, ale nenastartované vlákno
  - může být nastaveno více vláken
    - začnou se provádět v "nějakém" pořadí
  - daemon vlákna běží i během ukončování JVM
  - nastavená vlákna se nevykonají, pokud JVM byla ukončena pomocí halt()
- `boolean removeShutdownHook(Thread hook)`
  - odstraní dříve nastavené vlákno
  - vrací false, pokud dané vlákno nebylo nastaveno

Runtime

- `Process exec(String command)`
  - spustí externí proces
  - několik variant funkce (různé parametry)
  - nemusí vždy spolehlivě fungovat
- třída `Process`
  - reprezentuje externí proces
  - metody
    - `void destroy()`
      - zabije proces
    - `int exitValue()`
      - návratová hodnota
    - `int waitFor()`
      - čeká, dokud proces neskončí
      - vrací návratovou hodnotu procesu
      - může být přerušen

java.lang.StringBuffer
java.lang.StringBuilder

Přehled

- "měnitelný" řetězec
  - instance třídy `String` jsou neměnitelné
- nejsou potomky `String`
  - `String`, `StringBuffer`, `StringBuilder` jsou final
- `StringBuffer`
  - bezpečný vůči vláknům
- `StringBuilder`
  - není bezpečný vůči vláknům
  - od JDK 5
- mají stejné metody
  - vše pro `StringBuffer` platí i pro `StringBuilder`

Přehled

- operátor + na řetězcích je implementován pomocí třídy `StringBuffer`

  výraz `x = "a" + 4 + "c"`

  je přeložen do

```
x = new
  StringBuffer().append("a").append(4).
  append("c").toString()
```

- základní metody – `append` a `insert`
  - definovány pro všechny typy

Konstruktory

- `StringBuffer()`
  - prázdný bufer
- `StringBuffer(String str)`
  - bufer obsahující str
- `StringBuffer(int length)`
  - prázdný bufer s iniciální kapacitou length
    - kapacita je během práce s bufrem dle potřeby zvětšována
- `StringBuffer(CharSequence chs)`
  - `CharSequence`
    - interface
    - implementují ho String, StringBuffer, StringBuilder,...

Metody

- `StringBuffer append(`*`typ`*` o)`
  - definována pro všechny primitivní typy, Object, String a StringBuffer
  - převede parametr na řetězec a připojí na konec
  - vrací referenci na sebe (this)
- `StringBuffer insert(int offset,` *`typ`* `o)`
  - definována pro všechny typy jako append
  - vloží řetězec na danou pozici
  - offset musí být >=0 a < aktuální délka řetězce v bufru
- `StringBuffer replace(int start, int end, String str)`
  - nahradí znaky v bufru daným řetezcem
- `StringBuffer reverse()`
  - převrátí pořadí znaků v bufru

Metody

- `StringBuffer delete(int start, int end)`
  - odstraní znaky z bufru
  - start, end – indexy do bufru
- `StringBuffer deleteCharAt(int i)`
  - odstraní znak na dané pozici
- `char charAt(int i)`
  - znak na dané pozici
- `int length()`
  - aktuální délka řetězce v bufru
- `String substring(int start)`
- `String substring(int start, int end)`
  - vratí podřetezec

java.lang.Math

java.lang.Math

- statické atributy a metody pro základní matematické konstanty a operace
- atributy
  - PI, E
- metody
  - abs, ceil, floor, round, min, max,...
  - pow, sqrt,...
  - sin, cos, tan, asin, acos, atan,...
  - toDegrees, toRadians,...
  - ...

JAVA

Kolekce

Přehled

- (collections)
- kolekce ~ objekt obsahující jiné objekty
- např. – pole
  - jen pole nestačí
    - mnoho výhod (vestavěný typ, rychlý přístup, prvky i primitivní typy, ...
    - omezení – např. pevná velikost
- Java collection library
  - sada interfaců a tříd poskytujících dynamická pole, hašovací tabulky, stromy, ...
  - součást balíku **java.util**
    - ne vše v java.util je kolekce

Kolekce a JDK 5

- JDK < 5
  - prvky kolekcí – typ **Object**
    - nelze vkládat primitivní typy
- JDK 5
  - kolekce pomocí generických typu
    - fungují kolekce i bez <> - "raw" types
  - stále nelze pro primitivní typy
    - List<int> - chyba při překladu
  - metody zůstaly „stejné“

Ještě k polím: java.util.Arrays

- `java.util.Arrays`
  - sada metod pro práci s poli
  - součást knihovny kolekcí
- metody
  - všechny jsou statické
  - většinou definovány pro všechny primitivní typy a pro Object
- `int binarySearch(`*`typ`*`[] arr, `*`typ`*` key)`
  - hledání prvku v poli
  - binární vyhledávání
  - pole musí být setříděno vzestupně
  - vrací index prvku pokud v poli je nebo zápornou hodnotu indexu, kam by prvek patřil, kdyby v poli byl

Ještě k polím: java.util.Arrays

- `boolean equals(`*typ*`[] a1, `*typ*`[] a2)`
  - porovnává, zda jsou pole stejná, tj. stejně dlouhá a obsahují stejné prvky
  - prvky jsou stejné, pokud `(e1==null ? e2==null : e1.equals(e2))`
- `void fill(`*typ*`[] arr, `*typ*` val)`
  - vyplní všechny prvky pole parametrem `val`
- `void fill(`*typ*`[] arr, int from, int to, `*typ*` val)`
  - vyplní zadanou část pole parametrem `val`
- `void sort(`*typ*`[] arr)`
  - setřídí pole vzestupně
  - quicksort pro primitivní typy, mergesort pro Object
- `void sort(`*typ*`[] arr, int from, int to)`
  - setřídí zadanou část pole

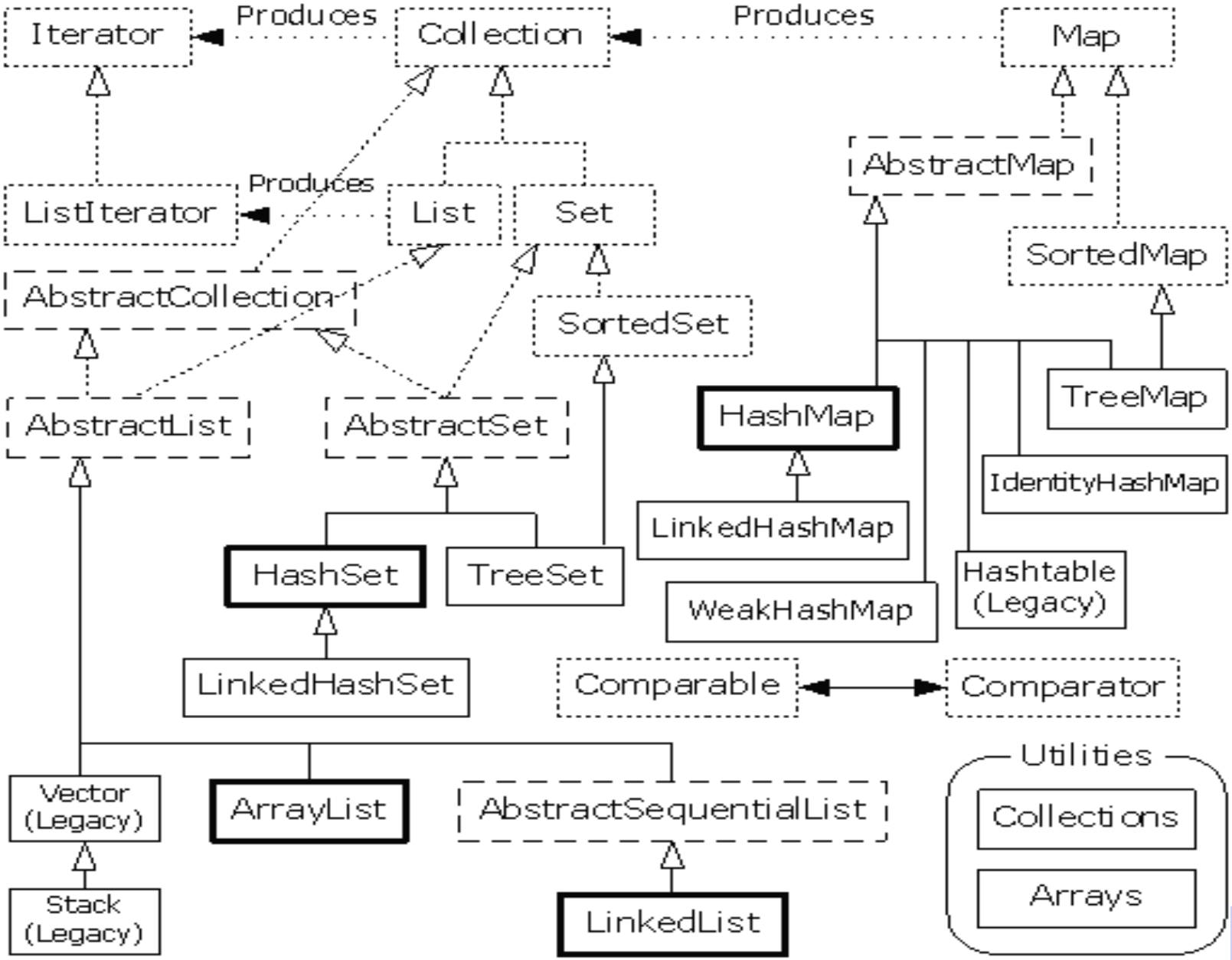
Třídění pole

- `void sort(Object[] arr)`
  - prvky pole musí být porovnatelné, tj. implementovat interface java.lang.Comparable<T>
    - metoda int compareTo(T o)
- `void sort(T[] arr, Comparator<? super T> c)`
  - prvky stále musí být porovnatelné
  - na porovnávání se použije objekt c
  - interface java.util.Comparator<T>
    - int compare(T o1, T o2)
  - pro vyhledávání
    - int binarySearch(T[] a, T key, Comparator<? super T> c)

Základní kolekce

- dva základní druhy – interface **Collection** a **Map**
- **Collection<E>**
  - skupina jednotlivých prvků
  - **List<E>**
    - drží prvky v nějakém daném pořadí
  - **Set<E>**
    - každý prvek obsahuje právě jednou
- **Map<K,V>**
  - skupina dvojic klíč–hodnota
- pro každý druh kolekce existuje alespoň jedna implementace
  - většinou více

Hierarchie kolekcí (JDK 1.4)



Hierarchie kolekcí

- kolekce neimplementují přímo daný interface
- implementují třídy AbstractSet, AbstractList, AbstractMap
  - abstraktní třídy
  - poskytují základní funkčnost dané kolekce
  - každá implementace interfacu Set, List, Map by měla rozšiřovat danou Abstract třídu
- používání kolekcí
  - obvykle přes interface daného druhu kolekce
  - př. `List c = new ArrayList()`
  - lze potom v aplikaci snadno vyměnit implementaci

Iterator<E>

- kolekce nemusí přímo podporovat přístup k prvkům
- kolekce mají metodu
  - Iterator<E> iterator()
  - vrací objekt typu Iterator<E>, který umožní projít všechny prvky v kolekci
- metody
  - `E next()`
    - vrací další prvek kolekce
  - `boolean hasNext()`
    - true, pokud jsou další prvky
  - `void remove()`
    - odstraní poslední vrácený prvek z kolekce

Iterator<E>

- implementace iteratoru a vztah k prvkům kolekce záleží na konkrétní kolekci

```
List c = new ....
...
Iterator e = c.iterator();
while (e.hasNext()) {
  System.out.println(e.next());
}
```

- nový cyklus **for** na kolekce s iteratorem

```
for (x:c) {
  System.out.println(x);
}
```

Collection<E>

- `boolean add(E o)`
  - přidá objekt do kolekce
  - vrací false, pokud se nepodařilo objekt přidat
  - volitelná metoda
- `boolean addAll(Collection<? extends E> c)`
  - přidá všechny prvky
  - vrací true, pokud přidala nějaký prvek
  - volitelná metoda
- `void clear()`
  - odstraní všechny objekty
  - volitelná metoda
- `boolean contains(E o)`
  - vrací true, pokud je objekt v kolekci

Collection<E>

- `boolean containsAll(Collection<?> c)`
  - vrací true, pokud jsou všechny dané objekty v kolekci
- `boolean isEmpty()`
- `Iterator<E> iterator()`
- `boolean remove(E o)`
  - vrací true, pokud odstranila objekt z kolekce
  - volitelná metoda
- `boolean removeAll(Collection<?> c)`
  - snaží se odstranit dané objekty z kolekce
  - vrací true, pokud něco odstranila
  - volitelná metoda
- `boolean retainAll(Collection<?> c)`
  - odstraní objekty, které nejsou v `c`
  - volitelná metoda

Collection<E>

- `int size()`
  - počet prvků v kolekci
- `Object[] toArray()`
  - vrátí pole obsahující všechny prvky kolekce
- `T[] toArray(T[] a)`
  - vrátí pole obsahující všechny prvky kolekce
  - vrácené pole je stejného typu jako je pole `a`

```
List<String> c;
....
String[] str = c.toArray(new String[1]);
```

List<E>

- potomek Collection
- udržuje prvky v nějakém pořadí
- může obsahovat jeden prvek vícekrát
- má metodu `E get(int index)`
  - vrátí prvek na dané pozici v kolekci
- kromě Iteratoru umožňuje získat i ListIterator
- ListIterator
  - potomek iteratoru
  - umožňuje
    - procházet prvky i v obráceném pořadí – metody `previous()`, `hasPrevious()`
    - přidávat a nahrazovat prvky – metody `add()`, `set()`

List<E>

- dvě implementace
- **ArrayList**
  - implementován polem
  - rychlý náhodný přístup k položkám
  - pomalé přidávání doprostřed
- **LinkedList**
  - rychlý sekvenční přístup
  - pomalý náhodný přístup
  - má navíc metody
    - addFirst()
    - removeFirst()
    - addLast()
    - removeLast()
    - getFirst()
    - getLast()

Set<E>

- potomek Collection
- nepřidává žádnou novou metodu
- každý prvek může obsahovat pouze jednou
- několik implementací
- **HashSet**
  - velmi rychlé vyhledání prvku v kolekci
  - nedodržuje pořadí
- **TreeSet**
  - Set implementovaný pomocí červeno-černých stromů
  - implementuje **SortedSet**
    - prvky jsou setříděny
    - umožňuje vrátit část (podmnožinu) kolekce
- **LinkedHashSet**
  - jako HashSet, ale udržuje pořadí prvků (pořadí vkládání)

Map<K,V>

- není potomek **Collection**
- kolekce dvojic klíč–hodnota
  ~ asociativní pole
- každý klíč obsahuje pouze jednou
- metody
  - `V put(K key, V value)`
    - asociuje klíč s hodnotou
    - vrací původní hodnotu asociovanou s klíčem (null, pokud klíč v kolekci nebyl)
  - `V get(K key)`
    - vrací hodnotu asociovanou s klíčem
  - `boolean containsKey(Object key)`
  - `boolean containsValue(Object val)`
  - `Set<K> keySet()`
  - `Collection<V> values()`

Map<K,V>: implementace

- několik implementací
- **HashMap**
  - implementace pomocí hašovací tabulky
  - konstantní čas přidávání a vyhledávání
- **LinkedHashMap**
  - jako HashMap
  - navíc při iterování dodržuje pořadí (pořadí přidávání nebo LRU)
  - o něco pomalejší
    - iterování je rychlejší
- **TreeMap**
  - implementace pomocí červeno-černých stromů
  - implementuje interface SortedMap
    - prvky jsou setříděny

HashMap<K,V>

- prvky musí správně implementovat metody `hashCode()`
- dva objekty, které jsou stejné (ve smyslu metody equals) musí vracet stejný hashCode
- různé objekty nemusí nutně vracet různý hashCode
- používá se hašování s řetězci
  - různé objekty se stejným hashCode budou ve stejném řetězci
- HashMap má na začátku nějaký počet "políček", do kterých se hašuje = kapacita
- faktor využití = počet prvků / kapacita
- při dosažení daného faktoru (implicitně 0.75) se kapacita zvětší a tabulka se "přehašuje"
  - kvůli rychlosti přístupu

Třída Collections

- obdoba třídy Arrays
- sada statických metod pro práci s kolekcemi
- metody
  - binarySearch
  - fill
  - sort
  - rotate
  - shuffle
  - reverse
  - ...

Nemodifikovatelné kolekce

- metody na Collections
  - unmodifiableCollection
  - unmodifiableList
  - unmodifiableSet
  - unmodifiableMap
- mají jeden parametr (daný druh kolekce)
- vrací "read-only verzi" kolekce, která obsahuje stejné položky jako dodaná kolekce

Synchronizace

- většina kolekcí není bezpečná vůči vláknům
- bezpečné (synchronizované) kolekce se vytvářejí stejně jako nemodifikovatelné
- metody na na Collections
  - synchronizedCollection
  - synchronizedList
  - synchronizedSet
  - synchronizedMap

"Staré" kolekce (Java 1.0, 1.1)

- Java collection library od verze Javy 1.2 přetvořena (List, Set, Map)
- původní kolekce
  - neměly by se používat
    - ale občas se použití nelze vyhnout
  - byly zahrnuty do nové verze (tj. také implementují List, Set nebo Map)
- Vector
  - obdoba ArrayList
- Enumeration
  - obdoba Iterator
- Hashtable
  - obdoba HashMap
- ...