Tvorba objektů

- **Objekty lze vytvářet, operovat nad nimi, ukládat je ve vnější paměti, přenášet je či využívat telekomunikačně pouze v čase běhu (run-time).**

- **Objekt (čili instance třídy) se vytváří v operační paměti na tzv. haldě. Obsahuje nestatické čili instanční atributy primitivních i referenčních typů. Objekt nemusí být kontinuální v paměti.**

- **Nový objekt se vytváří konstruktorem a event. nestatickými inicializátory.**

- **Každý objekt referuje jedinečný class-object popisující jeho třídu. Onen obsahuje statické čili třídní atributy a reference ke kódům metod, konstruktorů a inicializátorů. Kódy existují jen v jediném exempláři, neb jsou tzv. reentrantní (tj. neobsahují proměnná data) a tudíž dle nich může postupovat více vláken současně.**

- **JVM vede evidenci vytvořených objektů a referencí, které na ně odkazují.**

- **Nereferované objekty ruší při nedostatku paměti garbage collector. Ten lze programově zavolat metodou System.gc( ) či Runtime.gc( ) ;**

Konstruktory

Konstruktor vytváří na haldě objekt a dle parametrů nastavuje jeho atributy.

Připomíná konkrétní nestatickou metodu (funkci), která:

- **má jméno stejné jako třída.**
- **nevyznačuje návratový typ, tj. ani void, neboť vrací právě referenci svého typu na objekt, který vytvořil.**
- **má modifikátor viditelnosti (i private).**
- **není členem třídy, nedědí se a nemůže být abstraktní ani finální.**
- **konstruktory lze přetěžovat.**

- **Každá třída - tedy i abstraktní - má alespoň jeden konstruktor.**

- **Není-li ve třídě žádný konstruktor definován, vytvoří se skrytý implicitní konstruktor bez parametrů, který volá pouze konstruktor přímého předka bez parametrů:**

```
public JménoTřídy ( ) { super( ); }
```

Je-li však nějaký konstruktor explicitně definován, pak se implicitní nevytvoří a eventuálně požadovaný konstruktor bez parametrů je nutno definovat explicitně – případně i s neprázdným tělem.

Vzájemné volání konstruktorů

Pro začátek těl konstruktorů platí tato pravidla pro tři případy:

1. `public JménoTřídy ( ... ) { super(…); … }`
   Zprvu se volá příslušný existující konstruktor přímého předka.

2. `public JménoTřídy ( ... ) { this(…); … }`
   Zprvu se volá existující přetížený konstruktor téže třídy.
   Tuto fintu lze použít i několikrát, avšak nesmí dojít k cyklu.

3. `public JménoTřídy ( ... ) { … }`
   `public JménoTřídy ( ... ) { super( ); … }`

   } **jsou ekvivalentní**

   Začíná-li jinak než v případech 1 a 2, skrytě se na začátek přidá volání `super( );` tj. konstruktoru přímého předka bez parametrů.
   Neexistuje-li onen, kompilátor ohlásí chybu.

Vyhodí-li konstruktor či nestatický inicializátor výjimku, objekt nevznikne.

Vytvoření objektu

Objekt vytváří konstruktor instanciací třídy jedním z těchto způsobů:

- **aktivací konstruktoru pomocí klíčového slova:**

  new Type(…) resp. new Type(…) { ... }

  - kde Type je jméno třídy resp. interfejsu

  pro pole:

  new Type[num] resp. new Type[ ] { ..., ..., ... }

  - kde Type je jméno třídy, interfejsu nebo primitivní typ.

  Každé pole (i primitivů) lze referovat též typem java.lang.Object.

- **klonováním pomocí generální metody clone( )**

- **natažením ze serializovaného proudu**

- **nestatickou metodou java.lang.Class.newInstance( ) - např. takto:**

```
Class c = Class.forName( "ÚplnéJménoTřídy" );
Object x = c.newInstance( );
ÚplnéJménoTřídy z = ( ÚplnéJménoTřídy ) x;
```

Inicializace nestatických atributů

Nestatické atributy se inicializují při konstrukci každého objektu. Nejdříve dle společného pořadí deklarací a nestatických inicializátorů a pak v pořadí dokončovaných těl konstruktorů.

Nestatické čili instanční inicializátory připomínají metody, které mají přístup ke nestatickému i statickému kontextu, nic nevracejí, nemají signaturu, nevyznačují vyhazování výjimek, nelze je volat, nejsou členy třídy a nedědí se. Lze jich deklarovat i více.

Syntakticky jsou to pouhé bloky: { ... }

Příklad:

```
final static byte MARK = 0xff;
byte[ ]  a = new byte [ 256 ] ;
{  for ( int i = 0; i < a.length; i++ )  a[ i ] = ( byte ) i;  }
{  a[ 100 ] =  MARK; }
```

Postup konstrukce

Konstrukce objektu je tím složitější proces, čím je jeho třída dále od kořene a čím je její bohatství větší.

Po zavolání konstruktoru se:

- **Alokuje paměť a inicializují se nestatické atributy na defaultní hodnoty.**
- **Postupně volají konstruktory všech nadtříd (směrem nahoru) až k java.lang.Object a to příkazem super(…) volající některý existující konstruktor příslušné signatury.**
- **Pak se (směrem dolů) postupně dle každé třídy re-inicializují nestatické atributy (případně i nestatickými inicializátory) a teprve pak se dokončí těla jejich volaných konstruktorů.**

Proces konstrukce se může i stromově větvit - např.:

```
int i = 100, j = i+10, w = 300, h = w/2 ;
Rectangle  r = new  Rectangle( new Point( i, j ),  new Dimension( w, h ) );
```

Postup konstrukce

Těla konstruktorů a nestatických inicializátorů mohou obsahovat definice, příkazy a také volání statických i nestatických metod.

Upozornění:

Je-li nestatická metoda přepsána v potomkovi a je volána v konstruktoru či nestatickém inicializátoru předka, pak se vezme metoda potomka.

Tomuto jevu lze zabránit:

- **metoda předka se skryje přísnějším přístupovým modifikátorem – (např. private), čímž se tato metoda nepřepíše.**
- **či použitím statické metody.**

Rozhraní (Interface) Comparable < T >

vyznačuje porovnávatelné objekty tím, že definuje metodu

public int compareTo (T that) ; kde T je parametrický typ (od v. 1.5),

která vrací >0, 0, <0 pro this > that, this == that, this < that

Lexikografické porovnání řetězů: int n = s1.compareTo(s2);

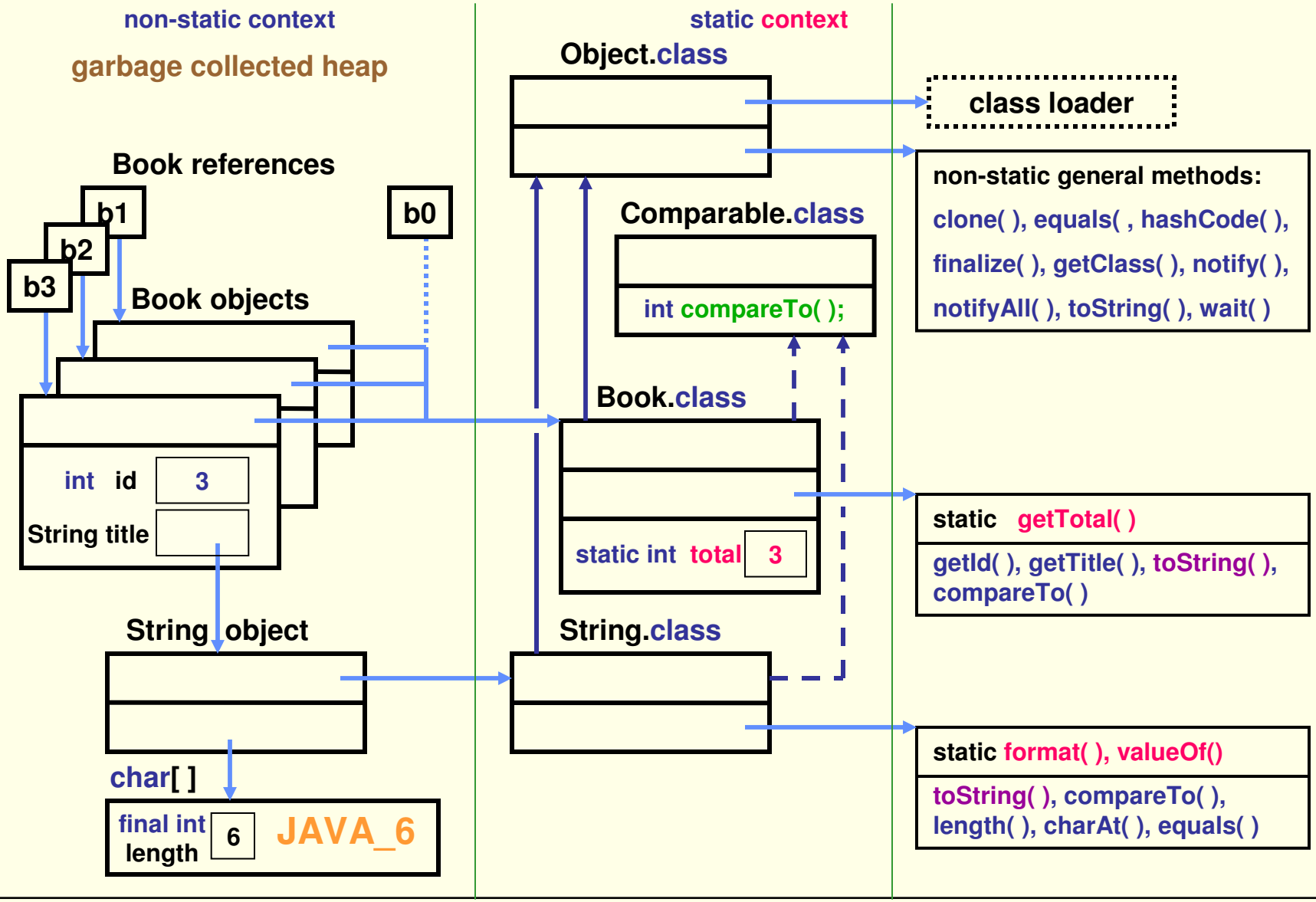
```
class Item implements Comparable {
    String  title;
    float price=0.0;
    public int compareTo( Object o ) {
        Item that = ( Item ) o;
        float diff = this.price – that.price;          // řazení vzestupně
        if ( diff == 0 ) return 0;
        return  diff > 0  ?  1 : -1 ;
    }
}
```

Příklad

```
class Book implements Comparable {
   private static int total;
   private int id;
   private String title;
   public Book( String title )  {  this.title = title;  id = ++total; }

   public static int getTotal( ) { return total; }
   public int getId( ) { return id; }
   public String  getTitle( ) { return title; }
   public int compareTo( Object o ) {
       if ( ! (o instanceof Book) ) System.out.println( "ClassCastException" );
       Book that = ( Book ) o;                          // can be thrown here
       return this.title.compareTo( that.title );
   }
   public String  toString( ) { return " Book: " +id+ " "+title; }
}  //  Inicializace vytvoří objekty a reference ( viz dále ):
Book b0 = null,  b1 = new Book("A"),  b2 = new Book("B"),
        b3 = new Book("JAVA_6") ;
```

Třídy, metody, atributy, objekty, reference



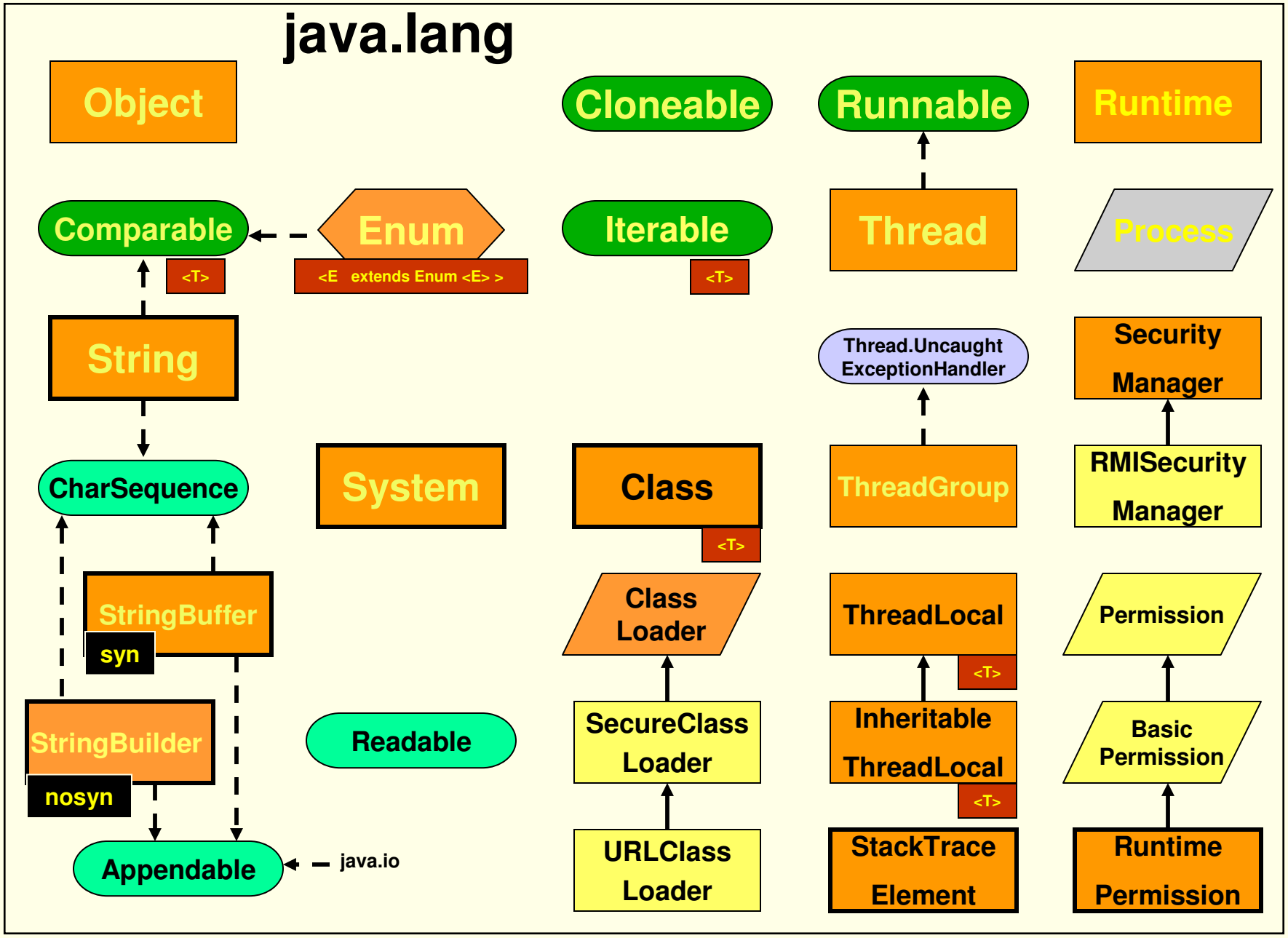
Balíček java.lang

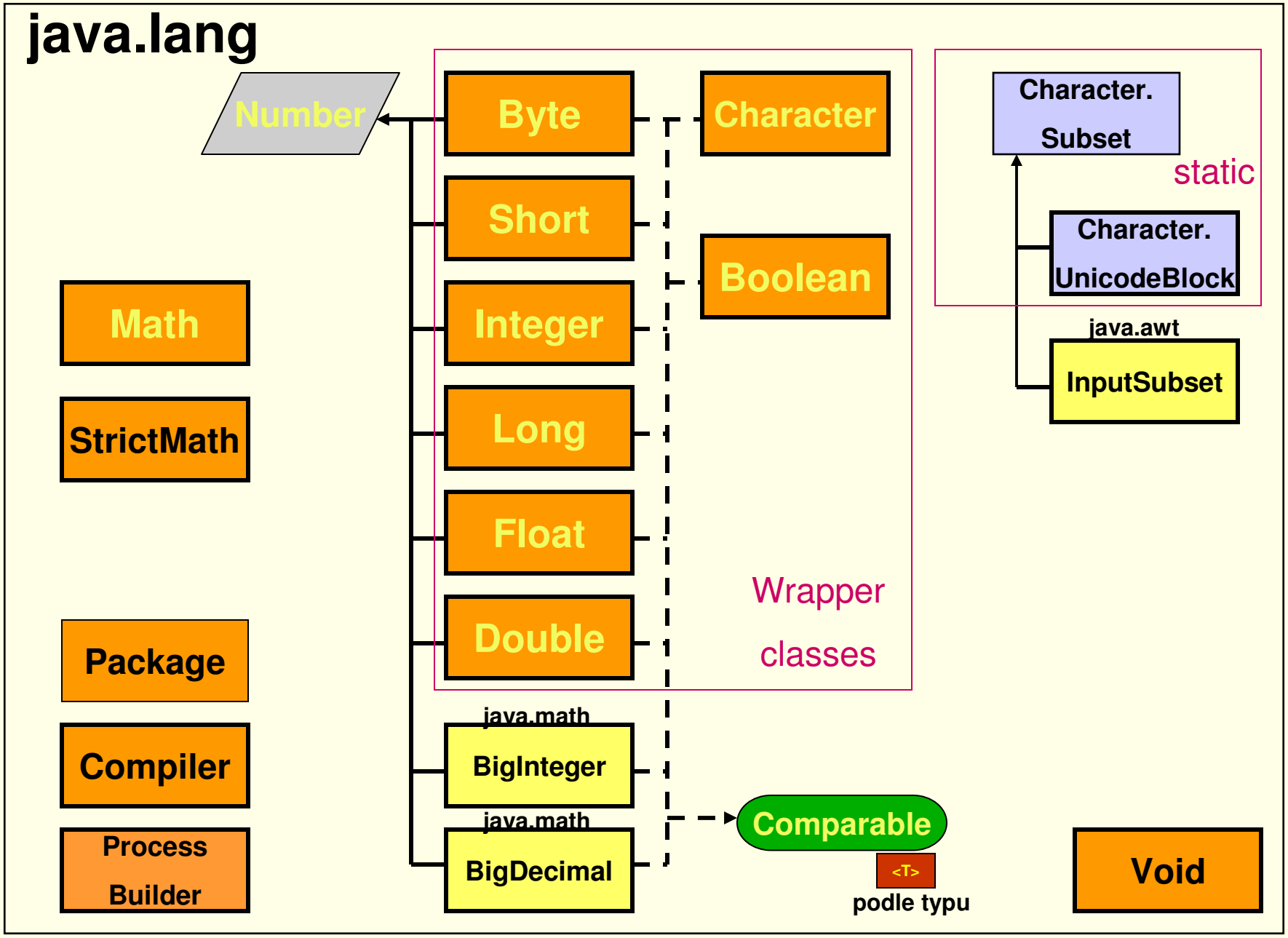
- Tento balíček je základní a nutný pro fungování Javy.
- Je defaultní – takže netřeba psát `import java.lang.*;`

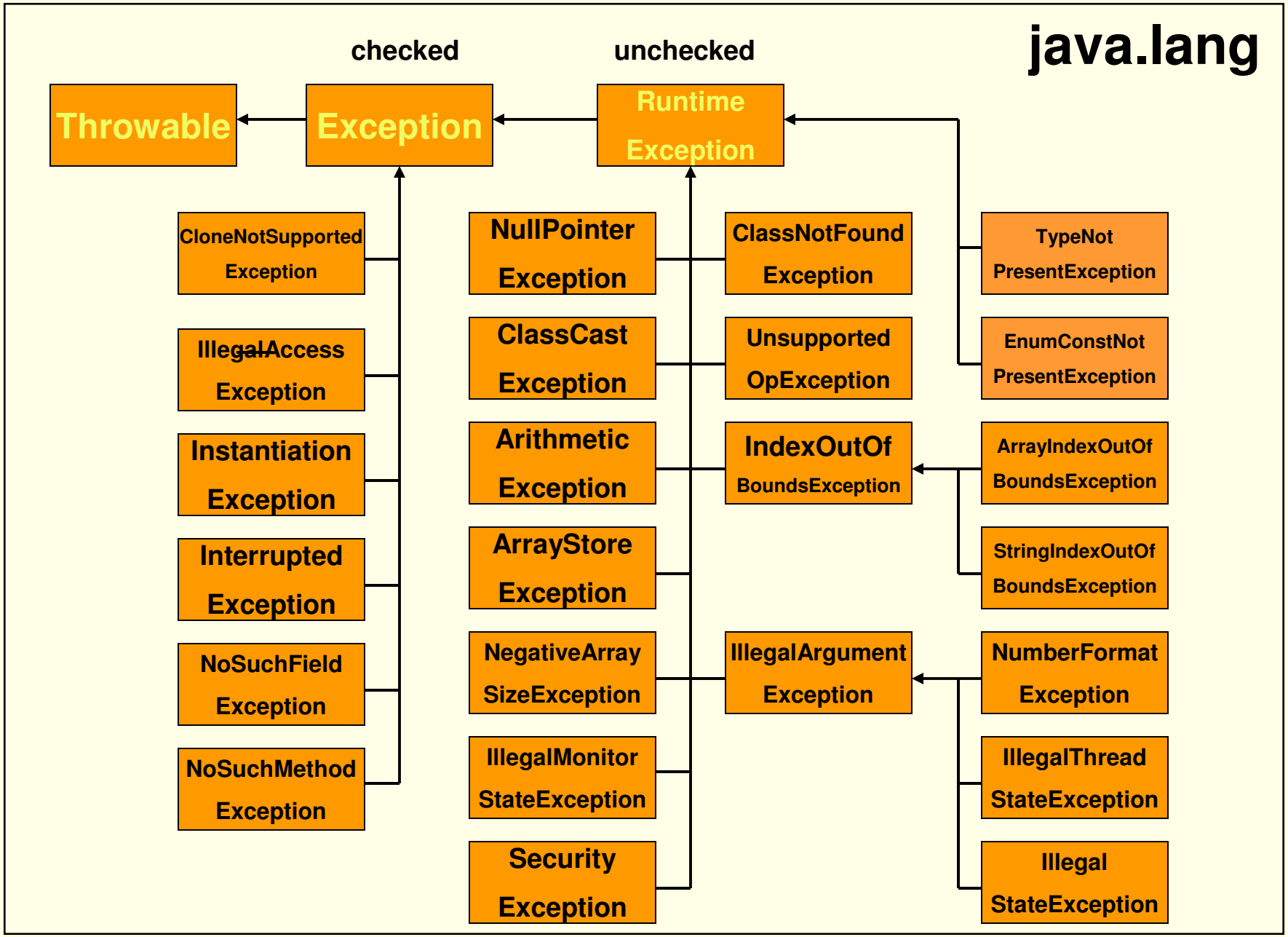
- Obsahuje základní interfejsy, třídy, Enum, Errors a Exceptions.

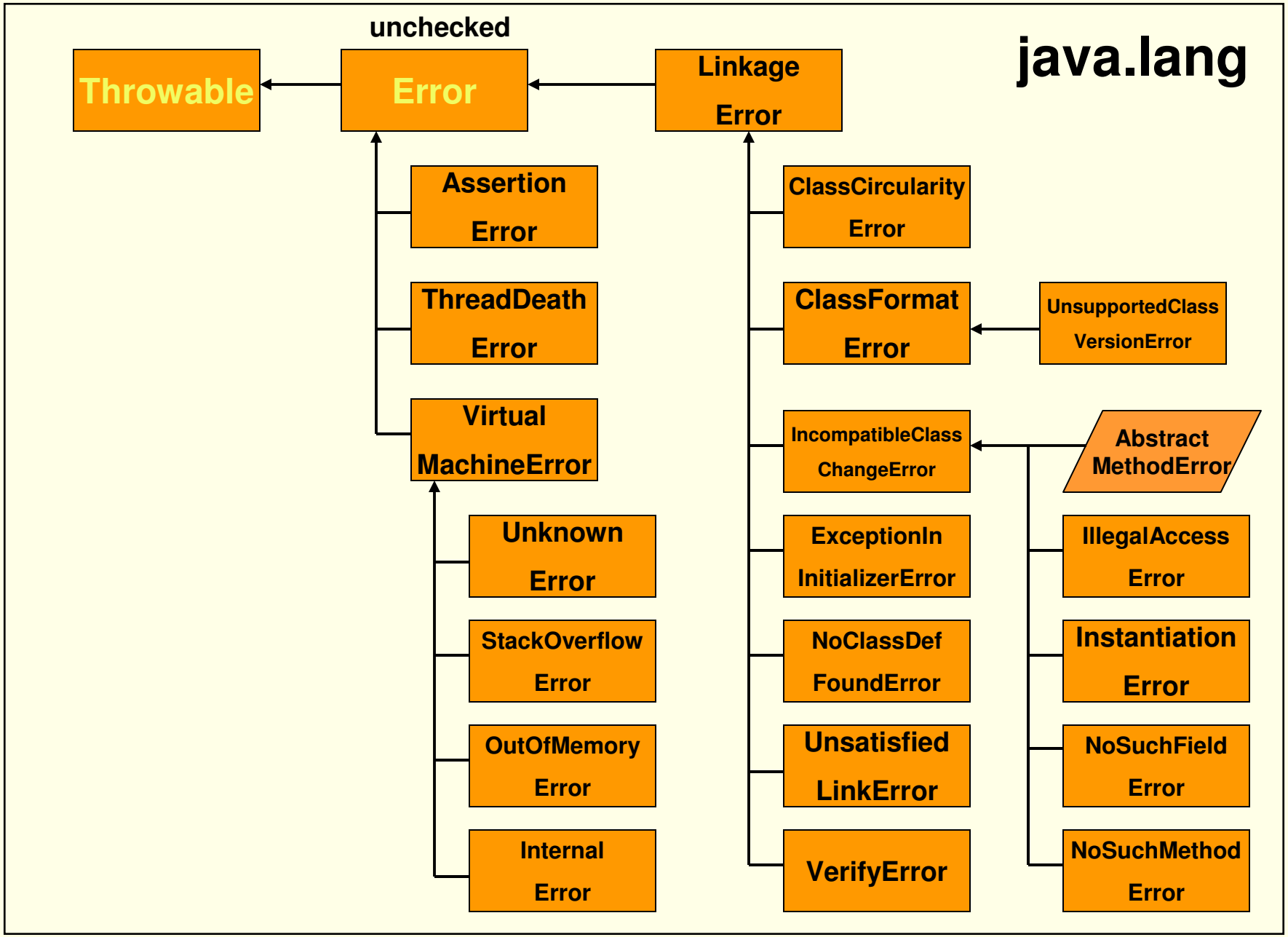
Podbalíčky:

- java.lang.annotation – pro podporu anotací (od v. 1.5)
- java.lang.instrument – pro redefinici tříd (od v. 1.5)
- java.lang.management – pro sledování a řízení JVM (od v. 1.5)
- java.lang.ref – pro práci s referencemi a garbage collectorem.
- java.lang.reflect – pro detailní práci s třídami a objekty.


java.lang
Object
Cloneable
Runnable
Runtime
Comparable
<T>
Enum
<E extends Enum <E> >
Iterable
<T>
Thread
Process
String
Thread.Uncaught ExceptionHandler
Security Manager
CharSequence
System
Class
<T>
ThreadGroup
RMISecurity Manager
StringBuffer
syn
Class Loader
ThreadLocal
<T>
Permission
StringBuilder
nosyn
Readable
SecureClass Loader
Inheritable ThreadLocal
<T>
Basic Permission
Appendable
java.io
URLClass Loader
StackTrace Element
Runtime Permission


java.lang
Number
Byte
Character
Short
Boolean
Integer
Long
Float
Double
Wrapper
classes
Character.
Subset
static
Character.
UnicodeBlock
java.awt
InputSubset
Math
StrictMath
Package
Compiler
Process
Builder
java.math
BigInteger
java.math
BigDecimal
Comparable
<T>
podle typu
Void


java.lang
checked
unchecked
Throwable
Exception
Runtime Exception
CloneNotSupported Exception
IllegalAccess Exception
Instantiation Exception
Interrupted Exception
NoSuchField Exception
NoSuchMethod Exception
NullPointer Exception
ClassCast Exception
Arithmetic Exception
ArrayStore Exception
NegativeArray SizeException
IllegalMonitor StateException
Security Exception
ClassNotFound Exception
Unsupported OpException
IndexOutOf BoundsException
IllegalArgument Exception
TypeNot PresentException
EnumConstNot PresentException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException
NumberFormat Exception
IllegalThread StateException
Illegal StateException


unchecked
java.lang
Throwable
Error
Linkage Error
Assertion Error
ThreadDeath Error
Virtual MachineError
Unknown Error
StackOverflow Error
OutOfMemory Error
Internal Error
ClassCircularity Error
ClassFormat Error
UnsupportedClass VersionError
IncompatibleClass ChangeError
Abstract MethodError
ExceptionIn InitializerError
NoClassDef FoundError
Unsatisfied LinkError
VerifyError
IllegalAccess Error
Instantiation Error
NoSuchField Error
NoSuchMethod Error

Třída java.lang.Object

- kořenem jediného stromu všech tříd. Nemá žádné dostupné atributy, má jen implicitní konstruktor a 11 přetížených, dostupných, instančních metod, které zdědí všechny další třídy.

- public boolean equals(Object obj) - definuje relaci ekvivalence objektů:
  x.equals(y) vrací true při identitě, tj. právě tehdy, když x = = y.
  Je-li tato metoda přepsána, je také nutno přepsat metodu hashCode pro dodržení uzance, aby ekvivalentní objekty vracely stejné hash kódy.

- public native int hashCode( ) - vrací číslo, které dle uzance:
  - vrací totéž číslo v jednom běhu - pokud se nezměnily žádné informace použité pro výpočet výsledku funkce equals( ).
    (Např. deserializovaný transient atribut zhatí kontrakt.)
  - ekvivalentní objekty - dle equals( ) - musejí vracet totéž číslo.
  - neekvivalentní objekty - dle equals( ) - nemusejí vracet různá čísla.
    Důsledek pro objekty téže třídy: hash1 != hash2 pak o1 != o2.
    Dosti rovnoměrně rozrůzněná čísla zlepšují funkci hešových tabulek.

Třída java.lang.Object

- **public final native Class<?> getClass( ) - vrací objekt typu Class, který popisuje jeho třídu a tak lze typ každého objektu přesně identifikovat.**
- **public String toString( ) - vydává popisný řetěz. Doporučuje se tuto metodu v potomcích přepsat tak, aby vracený popisný řetěz vyjadřoval typ a stav objektu. Metoda se automaticky volá při konkatenaci a v metodách print(...) či println(...).**
- **public native final void wait(...) throws InterruptedException - tyto metody umožňují zablokovat vlákno a zařadit ho do čekárny určitého objektu. A případně zadat maximální délku čekání.**
- **public native final void notifyAll( ) - uvolňuje všechna vlákna z čekání.**
- **public native final void notify( ) - uvolňuje některé vlákno z čekání.**

Třída java.lang.Object

- **protected native Object clone( ) throws CloneNotSupportedException - umožňuje mělké klonování potomků.**

- **protected void finalize( ) throws Throwable - tuto metodu volá garbage collector když hodlá objekt zrušit. Potomek může tuto metodu přepsat a provést:**
  - **nějaké úkony před zrušením objektu – např. získat jeho data. Metoda musí zavolat super.finalize( ).**
  - **zachránit tento objekt tím, že předá jeho referenci objektu, který je referován nějakým živým vláknem.**

Garbage collector lze vyvolat metodou System.gc( ) či Runtime.gc( ),
Sám se aktivuje při nedostatku požadované souvislé paměti
Neaktivuje se při ukončování aplikace.

Třída String

Řetězy tj. objekty typu String jsou imutabilní. Řetězové literály se ukládají do String Constant Poolu.

char charAt(int index) –
int compareTo(Object o) –
String concat(String s) –
boolean equals (Object o) –
boolean equalsIgnoreCase (String s) –
int length( ) –
int indexOf(String s)
int lastIndexOf(String s)
String[ ] split(String regex)
String replace(char old, char new)
String substring(int begin)
String substring(int begin, int end)
String trim(String s) –
String toUpperCase(String s)
String toLowerCase(String s)

Třídy StringBuffer a StringBuilder

Umožňují shodné operace s řetězy. StringBuffer je synchronizovaný, StringBuilder není – ale je však rychlejší a doporučuje se.
Oba nejsou Comparable a nemají přepsanou metodu equals.

```
char  charAt( int index ) –
StringBuilder append( … ) –
StringBuilder insert( int offset, … ) –
StringBuilder delete( int begin, int end ) –
int length( ) –
int indexOf( String s )
int lastIndexOf( String s )
String replace( char old, char new )
String reverse( )
String substring( int begin )
String substring( int begin, int end )
String trimToSize(  ) –
String toString( )
```

Třída System

má statické atributy: InputStream in, PrintStream out, PrintStream err

a dále jen statické metody:

void arraycopy(Object src, int spos, Object d, int dpos, int length) - kopie

long currentTimeMillis( ) – UTC čas v milisekundách

void exit(int code) – ukončení aplikace

void gc( ) – spustí garbage collector - skrze Runtime.gc( )

Map<String, String> getenv( ) - informace o prostředí

Properties getProperties( ) - vlastnosti prostředí

void setProperties(Properties props) - nastavení vlastnosti prostředí

void load(String filename)

void loadLibrary(String filename)

String mapLibraryName(String libname)

void setIn(InputStream is)

void setErr(PrintStream ps)

void setOut(PrintStream ps)

SecurityManager getSecurityManager( )