Ngôn ngữ lập trình Java
(Java Programming Language)

Môn học: Ngôn ngữ lập trình Java

Khoa: CNPM, Trường Đại học CNTT – ĐHQG TpHCM

GV: Th.S. Huỳnh Tuấn Anh

Email: anhht@uit.edu.vn

Nội dung môn học

Chương 1: Giới thiệu về NNLT Java

Chương 2: Hướng đối tượng trong Java

Chương 3: Các lớp tiện ích trong Java

Chương 4: Quản lý Exception

Chương 5: Nhập xuất

Chương 6: Xử lý đa luồng

Chương 7: Kết nối và thao tác CSDL với JDBC

Chương 8: Lập trình GUI với AWT & Swing

Chương 3. Các lớp tiện ích trong Java

- String/Number
- String builder/String buffer
- DateTime
- Enumerate
- Plan text file I/O
- Using Regular expression
- Random
- Generic
- Generic collection
- Annotation

STRING CLASS

String class (1/3)

```
String s1 = "uit.", s2 = " edu.vn ", s3, s4;
int i = s1.length();              // ?
boolean b = s1.isEmpty();         // ?
char c = s1.charAt(i - 1);        // ?
s3 = s1.concat(s2);               // ?
s4 = s1 + s2;                     // ?
b = s3 == s4;                     // ?
b = s3.equalsIgnoreCase(s4);      // ?
s3 = s4.substring(3);             // ?
s3 = s4.substring(5, 8);          // ?
s3 = s1 + s2;                     // ?
b = s3.contains(s1);              // ?
b = s3.endsWith(s2);              // ?
b = s3.startsWith(s1);            // ?
b = s3.startsWith("edu", 5);      // ?
```

String class (1/3)

```
String s1 = "uit.", s2 = " edu.vn ", s3, s4;
int i = s1.length();              // i = 4
boolean b = s1.isEmpty();         // false
char c = s1.charAt(i - 1);        // c = '.'
s3 = s1.concat(s2);               // "uit. edu.vn "
s4 = s1 + s2;                     // "uit. edu.vn "
b = s3 == s4;                     // false – String is an object
b = s3.equalsIgnoreCase(s4);      // true
s3 = s4.substring(3);             // ". edu.vn "
s3 = s4.substring(5, 8);          // "edu"
s3 = s1 + s2;                     // "uit. edu.vn "
b = s3.contains(s1);              // true
b = s3.endsWith(s2);              // true
b = s3.startsWith(s1);            // true
b = s3.startsWith("edu", 5);      // true
```

String class (2/3)

```
i = s3.indexOf('u');              // 0
i = s3.indexOf(s2);               // 4
i = s3.indexOf("u", 0);           // 0
i = s3.lastIndexOf("u", 2);       // 7

s4 = s3.replace("t. ", "t.");     // "?"
s4 = s3.trim();                   // "?"
String[] s5 = s3.split("[ .]");// regular expression {?}
s5 = s3.split("[u ]");            // {?}
s5 = s3.split("[u e]");           // {?}
char[] s7 = s3.toCharArray();     // {?}
s4 = s3.toUpperCase();            // "?"
s2 = s4.toLowerCase();            // "?"
```

DATE TIME OPERATOR

Date operators (1/2)

```
import java.util.*;
import java.text.SimpleDateFormat;
SimpleDateFormat df = new SimpleDateFormat(
                                          "yyyy-MM-dd hh:mm:ss.SSS");
GregorianCalendar cld1 = new GregorianCalendar();
// current date time
try {
   Date d = df.parse("2014-13-36 36:65:82.976");
   String s = df.format(d); // "2015-02-06 13:06:22.976"
   cld1.setTime(d);
} catch (ParseException e) {}
int year = cld1.get(Calendar.YEAR);              // 2015
int month = cld1.get(Calendar.MONTH);            // 02
boolean b = month == Calendar.JANUARY;           // false
int day = cld1.get(Calendar.DAY_OF_MONTH);       // 02
int dayw = cld1.get(Calendar.DAY_OF_WEEK);       // 06
b = dayw == Calendar.FRIDAY;                     // true
```

Date operators (2/2)

```
int hour = cld1.get(Calendar.HOUR);            // 04
int minute = cld1.get(Calendar.MINUTE);        // 06
int second = cld1.get(Calendar.SECOND);        // 22
int milisec = cld1.get(Calendar.MILLISECOND); // 976

GregorianCalendar cld2 = (GregorianCalendar)cld1.clone();
cld2.add(Calendar.YEAR, -1);
                              // same operator for other fields
   too
year = cld2.get(Calendar.YEAR);                // 2014
b = cld1.after(cld2);                          // true
b = cld1.before(cld2);                         // false
```

ENUMERATE

Simple Enum

```
// Declaration
enum WorkingDays {MONDAY, TUESDAY,
  WEDNESDAY, THURSDAY, FRIDAY}

// Using
WorkingDays wd = WorkingDays.TUESDAY;
switch (wd){…}
```

Complex enum

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6);
    // two members, correspond to two constants in enum elements
    private final double  mass;  //  in kilograms
    private final double radius; // in meters
    Planet(double mass, double radius) { // call automatically
        this.mass = mass;
        this.radius = radius;
    }
    public double mass() { return mass; }
    public double radius() { return radius; }
}
…
float mass = EARTH.mass()
…
for (Planet p: Planet.values()){…p.mass() … p.radius()… }
```

PLAN TEXT FILE I/O

Plan text file I/O

```
// Type
import java.io;
…
try{
   // File exist
   if (File.exists("a.txt")){
      // Open
      BufferedReader input = new BufferedReader(new  FileReader("a.txt"));
      BufferedWriter output = new BufferedWriter(new  FileWriter("b.txt"));
      String line;
      // Repeat access until end of input
      while ((line = input.readLine()) == null){
         output.write(line); output.newLine();
      }
      // close
      input.close(); output.close();
   }
} catch (IOException e){
   String msg = e.getMessage();
}
```

STRING/NUMBER CASTING

String/Number casting

```
// Each class in right hand side is called wrapper
// class of the corresponding primitive type
byte  b = Byte.parseByte("128");
                              // NumberFormatException
short s = Short.parseShort("32767");
int   x = Integer.parseInt("2");
int   y = Integer.parseInt("2.5");
                              // NumberFormatException
int   z = Integer.parseInt("a");
                              // NumberFormatException
long  l = Long.parseLong("15");
float f = Float.parseFloat("1.1");
double d = Double.parseDouble("2.5");
```

STRING BUILDER /STRING BUFFER

String builder/String buffer

```
StringBuilder sb = new StringBuilder("abc");
sb.append(" def");              // "abc def"
sb.delete(3, 5);                // "abcef"
sb.deleteCharAt(4);             // "abce"
sb.insert(3, " d");             // "abc de"
sb.replace(2, 4, " ghi");       // "ab ghide"
sb.reverse();                   // "edihg ba"
sb.setCharAt(5, 'j');           // "edihgjba"
// StringBuffer: thread safe version
// of StringBuilder
   => StringBuilder is faster
```

USING REGULAR EXPRESSION

Regular expression

```
import java.util.regex.*;

Pattern pattern = Pattern.compile("abc|def",

   Pattern.CASE_INSENSITIVE);
Matcher matcher = pattern.matcher("abcdef  fgdsfabclks");
while (matcher.find()) {
    String s = matcher.group();    // the pattern found
    int    i = matcher.start();    // start position
    i        = matcher.end();      // "end + 1" but not "end"
}
// Result: (abc 0 3), (def 3 6), (abc 12 15)
```

RANDOM CLASS

Random

```
Random rdm = new Random();
int i = rdm.nextInt(10); // a number from 0 to 9
i = rdm.nextInt();
 // equivalent to rdm.nextInt(Integer.MAX_VALUE)
long l = rdm.nextLong();
 // not full rang long number can be returned
 // cause of java seed is only 48 bits
byte[] bar = new byte[10];
rdm.nextBytes(bar);
 // bar now contains 10 byte random numbers
float f = rdm.nextFloat();     // from 0.0 to 1.0
double f = rdm.nextDouble();   // from 0.0 to 1.0
```

GENERIC TYPE

One type generic

```
class GenericType<T>{
   // T is a type representation, not a specific type
   private T aT;
   public T getMember(){return aT;}
   public void setMember(T newT){aT = newT;}
}
class A{}

// use generic class with specific type int
GenericType<int> gInt = new GenericType<int>();
gInt.setMember(5);
int i = gInt.getMember();

// use generic class with specific type A
GenericType<A> gA = new GenericType<A>();
gA.setMember(new A());
A a = gA.getMember();
```

Bounded generic type

```
class GenericType<T extends A>{
   // T is a type representation, not a specific type
   // A is a specific type
   private T aT;
   public T getMember(){return aT;}
   public void setMember(T newT){aT = newT;}
}
class A{}
class B extends A{}
class C{}

GenericType<A> gA = new GenericType<A>(); // OK
GenericType<B> gB = new GenericType<B>(); // OK too
GenericType<C> gA = new GenericType<C>(); // Error, C is not A
```

GENERIC COLLECTION

ArrayList: Input

```
class A{int i;}
A[] arA = new A[10];                    // Predefined capacity required
…
List<A> alA = new ArrayList<A>(); // No predefined capacity
boolean b = alA.isEmpty();              // true

A aA = new A(); aA.i = 1;
alA.add(aA);                                  // add new
b = alA.isEmpty();                            // false
alA.add(aA);                    // add new again, duplicate accepted

A aoA = new A(); aoA.i = 2;
alA.add(1, aoA);              // insert to the 2nd position, (1, 2, 1)
```

ArrayList: Output

```
int s = alA.size();  // 3

A outA = alA.get(2);
b = outA == aoA;
                     // true

outA = alA.get(3);   // error, out of range


alA.set(2, aoA);     // replace the 3rd position, (1, 2, 2)

int i = alA.indexOf (aoA); // 1
i = alA.lastIndexOf (aoA); // 2

for (A a: alA){System.out.println(a.i);}          // 1, 2, 2

alA.remove(1);       // remove the 2nd position, (1, 2)
```

ArrayList: Sort by Arrays

```
class A implements Comparable<A>{     // implement
   Comparable<T>
   int i;
   public int compareTo(A another){   // implement compareTo(T
   t)
        if (i == another.i) return 0;
        if (i < another.i) return -1;
        return 1;
   }
}

Object[] arA = alA.toArray();         // convert to array
Arrays.sort(arA);                     // using Arrays.sort
for (Object a: arA){
   A a1 = (A)a;                       // revert to original
   type
   System.out.println(a1.i);
}
```

HashMap: Input

```
class A{int i;}
HashMap<int, A> aMap = new HashMap<int, A>();
// Error, key must be an object type
HashMap<Integer, A> aMap = new HashMap<Integer, A>();
// use the hash code of key then no order is warranted
boolean b = aMap.isEmpty();                 // true

A aA = new A(); aA.i = 1;
aMap.put(1, aA);          // add new
b = aMap.isEmpty();       // false
int i = aMap.size();      // 1

aMap.put(1, aA);          // replace the older one
i = aMap.size();          // no new adding with the same key
```

HashMap: Output

```
b = aMap.containsKey(1);          // true
b = aMap.containsValue(aA);       // true

A oA = aMap.get(1);               // access by key
B = oA == aA;                     // true

oA = aMap.get(2);                 // oA = null

b = aMap.remove(1);               // access by key
```

ANNOTATION

Annotation

```
class A{
   public int doSmt();

   @Deprecated()        // Do not use the next method
   public int oldMethod(){}

   @SuppressWarnings("deprecation")
                        // Do not display the warning on
                        // the use of a deprecated method
   public int aMethod(){oldMethod();}
}
class B{
   @Override            // The next method overrides a base method
   public int doSmt();
}
```

Nội dung đã học

- Exception: handling with try/catch/final statement
- Enumerate: simple and valued
- Plan text file I/O: read and write text file
- String builder: used when string change frequently
- String buffer: thread safe version of String builder
- String/Number casting: string parse
- Regular expression: search
- Random: generate, using
- Generic: applied for any one specific type or a hierarchy of a type
- Generic collection: ArrayList, HashMap
- Annotation: Starts with @