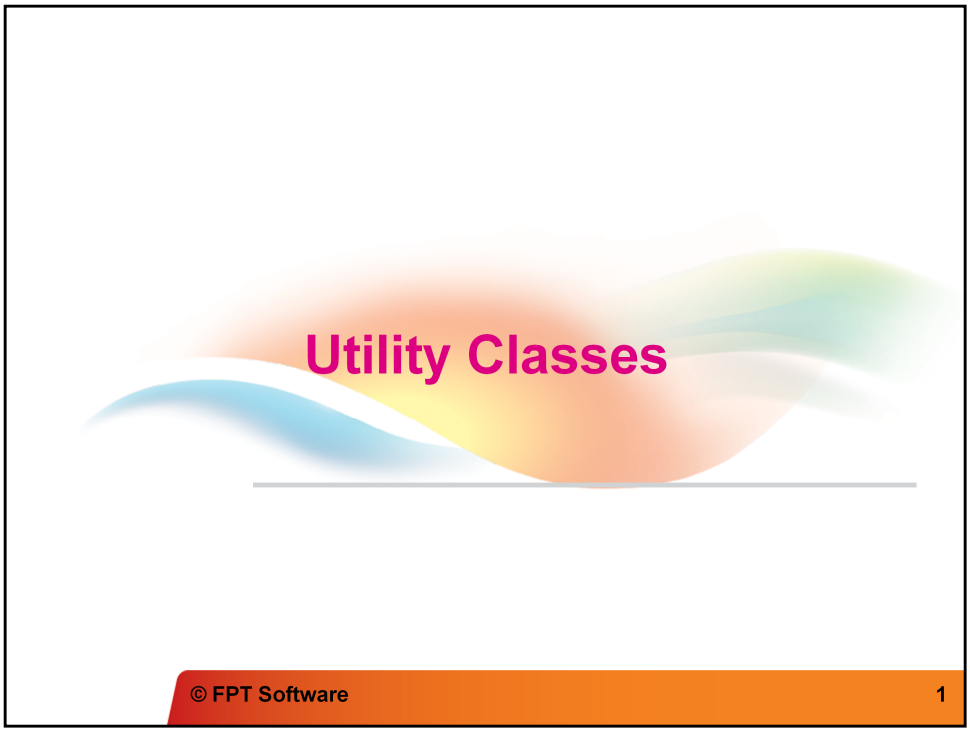


<*Lưu ý: không xóa nội dung cũ mà viết thêm lên trên như nhật ký*>

Tài liệu đào tạo: *<môn học, buổi học>*

Phiên bản tài liệu:

Người cập nhật:

Ngày cập nhật:

Tóm tắt nội dung cập nhật chính:

Ngày ban hành sử dụng:

Agenda

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

ENUMERATE

Simple Enum

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

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


Ask to search the infor about enum with value

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

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
```

STRONG TYPED/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();
```

Multi type parameter

```
class GenericType<T, U, V>{
   // Anny positive number of type parameter,
   private T aT;
   private U aU;
   private V aV;
   …
}
```

Bounded type generic

```
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
```

Multiple bounded type generic

```
class GenericType<T extends A1 & A2 & A3>{
   // T is a type representation, not a specific type
   // A1 is a specific type, class or interface
   // other A2, A3, … must be an interface
   private T aT;
   public T getMember(){return aT;}
   public void setMember(T newT){aT = newT;}
}
```

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)
```


equals

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
```


hashCode

Hashtable - HashMap

Vector – ArrayList

StringBuffer – StringBuilder

Collections.sync…(un safe)

HashMap: Output

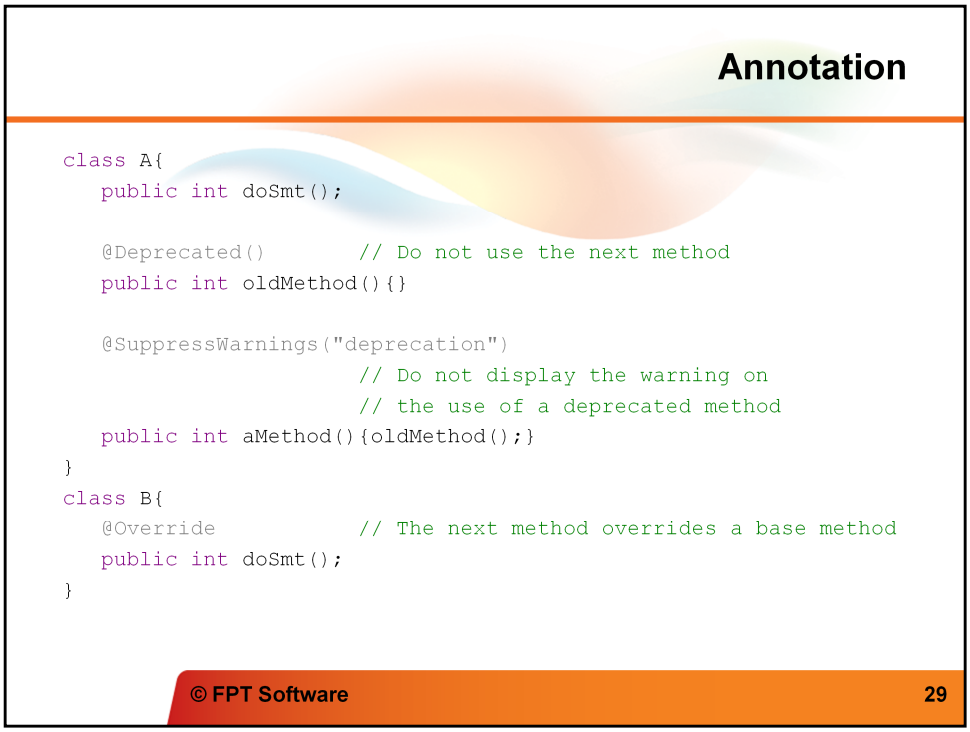
```
b = aMap.containsKey(1);                    // true
b = aMap.containsValue(aA);                 // true

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



Annotations, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.

Annotations have a number of uses, among them:

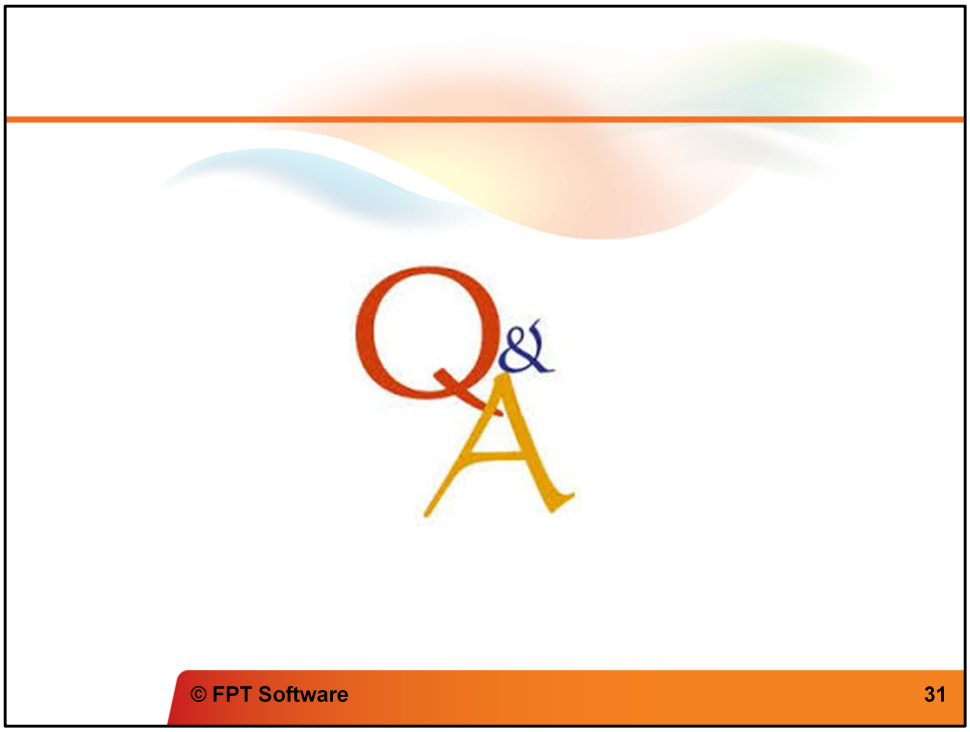
Information for the compiler — Annotations can be used by the compiler to detect errors or suppress warnings.

Compile-time and deployment-time processing — Software tools can process annotation information to generate code, XML files, and so forth.

Runtime processing — Some annotations are available to be examined at runtime.

Lesson summary

- 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 @


Q&A
© FPT Software
31