ГЕНЕТИЧНИ КЛАСОВЕ

ЛЕКЦИОНЕН КУРС "ПРОГРАМИРАНЕ НА JAVA"

Distributed @learning center

СТРУКТУРА НА ЛЕКЦИЯТА

- Общ стек
- Клас Object
- Класове-обвивки
- Смесени стекове
- Генетични класове
- Примери

ПРОБЛЕМ: СТЕК ЗА ЕЛЕМЕНТИ ОТ РАЗЛИЧЕН ТИП

- За char-елементи (сегашен вариант)

```
class Stack {
   private char[] stackElements;
   private int top;
      ...
   public void push(char x) {
      top++; stackElements[top] = x;
   }
   ...
```

Как да реализираме генерализация ?

Съществува ли елегантно решение ?

- За int-елементи
- За времена: клас 'Time'

→ 3 (обобщ. n) класа с общ код в по-голямата си част

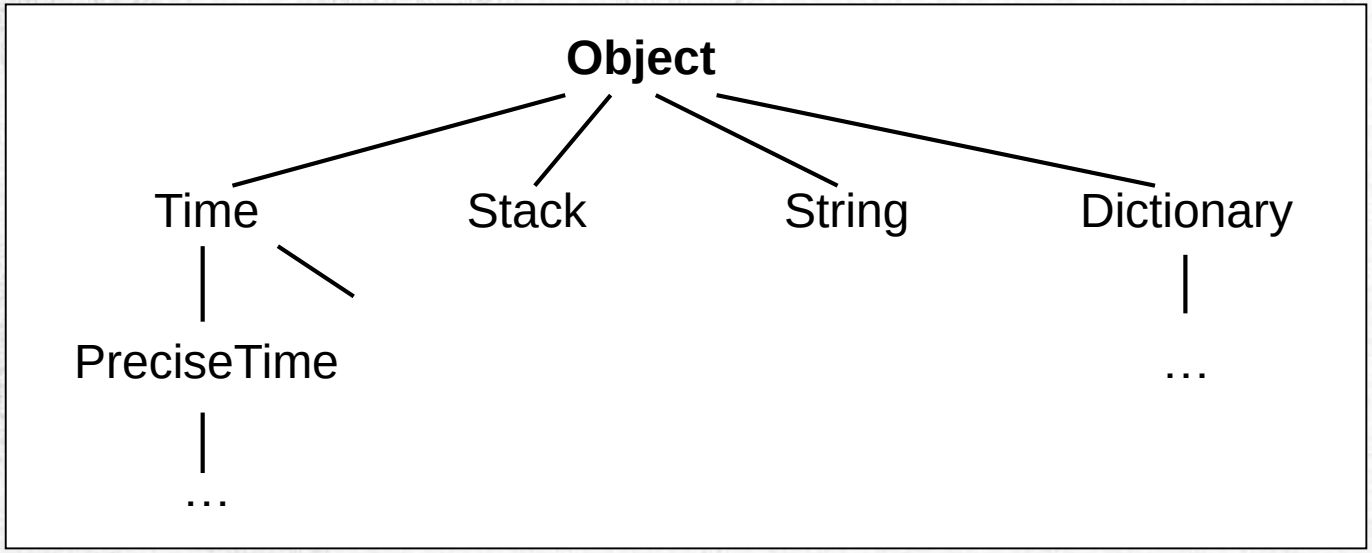
ОБЩ СТЕК: ТИП НА ЕЛЕМЕНТИТЕ 'OBJECT'

```
class Stack {
   private Object [] stackElements;
   private int top;

   public Stack(int n) {
      stackElements = new Object[n];
      top = -1;
   }

   public void push(Object x) {
      top++;
      stackElements[top] = x;
   }
   ...
}
```

ПОВТОРЕНИЕ: ПРАВИЛА ЗА СЪВМЕСТИМОСТ В ЙЕРАРХИЯТА НА КЛАСОВЕТЕ



Обектите на един подклас могат да стоят там, където се допускат обекти на суперкласа

```
Time t;
t = new PreciseTime(12, 10, 1);
t.printTime();
```

ОБЩ СТЕК:ИЗПОЛЗВАНЕ ПРИ 'TIME'

```
class Stack {
    private Object [] stackElements;
    private int top;

    public Stack(int n) {
        stackElements = new Object[n];
        top = -1;
    }

    public void push(Object x) {
        top++; stackElements[top] = x;
    }
    ...
}
```

```
Time t1 = new Time(8, 30);
Stack timeStack = new Stack(10);
timeStack.push(t1.copy());
t1.addMinutes(10);
timeStack.push(t1.copy());
t1.addMinutes(30);
timeStack.push(t1.copy());
```

Какво става при `timeStack.push(t1)`?

индиректно:
class Time extends Object

ОБЩ СТЕК: ИЗПОЛЗВАНЕ ПРИ 'CHAR'

```
class Stack {
     private Object [] stackElements;
     private int top;

     public Stack(int n) {
          stackElements = new Object[n];
          top = -1;
     }

     public void push(Object x) {
          top++; stackElements[top] = x;
     }
     ...
}
```

Wrapper класове

```
char ch = 'A';
Stack s = new Stack(10);
s.push(ch);
```

Очаква обект или подклас

```
Stackchar.java:52: Incompatible type for method.
Can't convert char to java.lang.Object.
```

WRAPPER-КЛАСОВЕ

Прост тип → клас

прост Тур ‘опакован' в клас

`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`double`	`Double`
`float`	`Float`
`boolean`	`Boolean`
`char`	`Character`
`void`	`Void`

в java.lang

За всички подкласове на java.lang.Object
→ могат да стоят там, където е разрешено Object.

WRAPPER-CLASS ‚CHARACTER‘

java.lang

Class Character

java.lang.Object
  java.lang.Character

All Implemented Interfaces:

Comparable, Serializable

---

public final class **Character**
extends Object
implements Serializable, Comparable

The `Character` class wraps a value of the primitive type `char` in an object. An object of type `Character` contains a single field whose type is `char`.

In addition, this class provides several methods for determining a character's category (lowercase letter, digit, etc.) and for converting characters from uppercase to lowercase and vice versa.

Character information is based on the Unicode Standard, version 3.0.

The methods and data of class `Character` are ... his file specifies various properties including name and general

The file and its description are available from

- http://www.unicode.org

Since:

1.0

See Also:

Serialized Form

Field Summary

static byte	**COMBINING_SPACING**
static byte	**CONNECTOR_PUNCTUA**
static byte	**CONTROL** General
static byte	**CURRENCY_SYMBOL**
static byte	**DASH_PUNCTUATION**
static byte	**DECIMAL_DIGIT_NUM**
static byte	**DIRECTIONALITY_ARABI**
static byte	**DIRECTIONALITY_BOUNDARY_NEUTRAL** Weak bidirectional character type "BN" in the Unicode specification.
static byte	**DIRECTIONALITY_COMMON_NUMBER_SEPARATOR** Weak bidirectional character type "CS" in the Unicode specification.
static byte	**DIRECTIONALITY_EUROPEAN_NUMBER** Weak bidirectional character type "EN" in the Unicode specification.
static byte	**DIRECTIONALITY_EUROPEAN_NUMBER_SEPARATOR** Weak bidirectional character type "ES" in the Unicode specification.
static byte	**DIRECTIONALITY_EUROPEAN_NUMBER_TERMINATOR** Weak bidirectional character type "ET" in the Unicode specification.
	…

The Character class wraps a value of the primitive type char in an object. An object of type Character contains a single field whose type is char.
In addition, this class provides several methods for determining a characters's category (lowercase letter, digit, etc.) and for converting characters from uppercase to lowercase and vice versa.

Constructor Summary

Character(char value)	Constructs a newly allocated Char

Method Summary

char	**charValue**() Returns the value of this Character object.
int	**compareTo**(Character ... Characte
int	**compareTo**(Object o) Compare
static int	**digit**(char ch, int radix) Returns
boolean	**equals**(Object obj) Compares this ob
static char	**forDigit**(int digit, int radix) Determines the character representation for a specific digit in the specified radix.
static byte	**getDirectionality**(char ch) Returns the Unicode directionality property for the given character.
static int	**getNumericValue**(c
static int	**getType**(char ch)
int	**hashCode**() Returns
static boolean	**isDefined**(char ch)
static boolean	**isDigit**(char ch) Determines if the specified character is a digit.
static boolean	**isIdentifierIgnorable**(char c
static boolean	**isISOControl**(char ch) De
static boolean	**isJavaIdentifierPart**(char ch
static boolean	**isJavaIdentifierStart**(char c
static boolean	**isJavaLetter**(char ch) **Deprecated.** *Replaced by isJavaIdentifierStart(char).*
static boolean	**isJavaLetterOrDigit**(char
static boolean	**isLetter**(char ch) Deter
static boolean	**isLetterOrDigit**(char ch)
static boolean	**isLowerCase**(char ch) Determines if the specified character is a lowercase character.
static boolean	**isMirrored**(char ch)
static boolean	**isSpace**(char ch) D
static boolean	**isSpaceChar**(char ch)
static boolean	**isTitleCase**(char ch)
	...

public Character(char value)

Constructs a newly allocated Character object that represents the specified char value.

public char charValue()

Returns the value of this Character object.

public static int getNumericValue(char ch)

Returns the int value that the specified Unicode character represents.

public static boolean isDigit(char ch)

Determines if the specified character is a digit.

public static boolean isLetter(char ch)

Determines if the specified character is a letter.

public static boolean isLowerCase(char ch)

Determines if the specified character is a lowercase character.

ОБЩ СТЕК: ИЗПОЛЗВАНЕ НА WRAPPER-КЛАС 'CHARACTER'

```
class Stack {
     private Object [] stackElements;
     private int top;

     public Stack(int n) {
          stackElements = new Object[n];
          top = -1;
     }

     public void push(Object x) {
          top++; stackElements[top] = x;
     }

     public Object top() {…}
     ...
}
```

```
Character ch;
Stack s = new Stack(10);
ch = new Character(Keyboard.readChar());
s.push(ch);
ch = (Character)s.top();
char c = ch.charValue();
```

Typ 'char'

'Character'

'char'

СЪВМЕСТИМОСТ И TYPE-CAST

```
class Stack {
    private Object [] stackElements;
    private int top;

    public Stack(int n) {
        stackElements = new Object[n];
        top = -1;
    }

    public void push(Object x) {
        top++; stackElements[top] = x;
    }

     public Object top() {…}
    ...
}
```

```
Character ch;
Stack s = new Stack(20);
ch = new Character(Keyboard.readChar());
s.push(ch);
ch = (Character)s.top();
char c = ch.charValue();
```

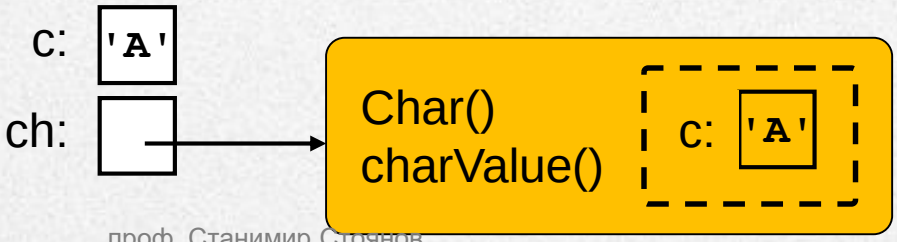
Правило за съвместимост:
обектите на един подклас могат да стоят там, където са допустими обекти на суперкласа.

Обратна посока (суперклас → подклас):
Typ-Cast на 'Object' към 'Character'

WRAPPER 'CHAR' ЗА 'CHAR': СОБСТВЕНА РЕАЛИЗАЦИЯ

```
class Char {
   private char c;
   public Char(char ch) {
      c = ch;
   }
   public char charValue() {
      return c;
   }
}
```

```
Char ch;
ch = new Char('A');
char c = ch.charValue();
```



ВИНАГИ “ЧИСТИ" STACKS: ЕДИН И СЪЩ БАЗОВ ТИП (НАПР. CHARACTER)?

Общ стек: тип на елементите 'Object'

```
class Stack {
   private Object [] stackElements;
   private int top;

   public Stack(int n) {
      stackElements = new Object[n];
      top = -1;
   }
   ...
}
```

До сега: Stacks от Time, Character, …

"СМЕСЕНИ" СТЕКОВЕ ВЪЗМОЖНИ

```
class Stack {
   private Object [] stackElements;
   private int top;

   public Stack(int n) {
      stackElements = new Object[n];
      top = -1;
   }
   ...
}
```

```
Stack mixedStack = new Stack(10);

mixedStack.push(new Char('A'));
mixedStack.push(new Time(8,30));
mixedStack.push(new Integer(20));
```

БИ БИЛО ДОБРЕ, АКО ИМА ТИП-ПАРАМЕТРИ … (1)

Вместо:

```
class Stack {
    private Object [] stackElements;
    private int top;

    public Stack(int n) {
        stackElements = new  Object [n];
        top = -1;
    }

    public void push(Object x) {
        top++; stackElements[top] = x;
    }

    public Object top() {
        . . .
    }
  . . .
}
```

БИ БИЛО ДОБРЕ, АКО ИМА ТИП-ПАРАМЕТРИ … (2)

сега:

```
class Stack  <T> {
    private T [] stackElements;
    private int top;

    public Stack(int n) {
        stackElements = new  T [n];
        top = -1;
    }

    public void push(T x) {
        top++; stackElements[top] = x;
    }

    public T top() {
       . . .
    }
  . . .
}
```

Т.е. произволен, но твърдо определен тип

… С АКТУАЛИЗИРАНЕ ПОСРЕДСТВОМ ТИП-АРГУМЕНТИ

```
Stack <Time> tStack;
Stack <Character> chStack;

tStack = new Stack <Time> ();
chStack = new Stack <Character> ();

tStack.push(new Time(1,20));
chStack.push(new Character('A'));
chStack.push(new Time(7,10));
```

С това: не са разрешени „смесени“ стекове

Грешка на типа: компилаторът очаква параметър от тип ‘Character’

ГЕНЕТИЧНИ КЛАСОВЕ: ДЕКЛАРАЦИЯ

Променлива на тип

```
class Pair <T> {
    private T first;
    private T second;

    public Pair(T fst, T scd) {
        first = fst;
        second = scd;
    }

    public T getFirst() {
      return first;
    }

    public T getSecond() {
      return second;
    }
  }
```

T: произволен, но определен тип

С това няма смесени двойки: напр. (Integer, String)

от Java 2 (Version 1.5)

ГЕНЕТИЧЕН ТИП: ПРОМЕНЛИВИТЕ НА ТИПОВЕ СЕ АКТУАЛИЗИРАТ

```
class BuildPairs {

  public static void main (String[] args) {
    Pair<Integer> pi;
    Pair<String>  ps;
    Integer i, j;

    i = new Integer(99);
    j = new Integer(100);
    pi = new Pair<Integer> (i, j);
    ps = new Pair<String> ("Hallo", "World");

    System.out.println(ps.getFirst() + " "
           + ps.getSecond());
    System.out.println(pi.getFirst().intValue()
           + " " + pi.getSecond().intValue());
  }
}
```

Генетичен тип

Аргумент-тип

ГЕНЕТИЧЕН ТИП: ИЗПОЛЗВАНЕ КАТО НОРМАЛЕН ТИП

```
class BuildPairs {

  public static void main (String[] args) {
    Pair<Integer> pi;
    Pair<String>  ps;
    Integer i, j;

    i = new Integer(99);
    j = new Integer(100);
    pi = new Pair<Integer> (i, j);
    ps = new Pair<String> ("Hello", "World");

    System.out.println(ps.getFirst() + " "
           + ps.getSecond());
    System.out.println(pi.getFirst().intValue()
           + " " + pi.getSecond().intValue());
  }
}
```

Задаване на променливи

Извикване на конструктор

Извикване на метод

```
% java BuildPairs

Hello World
99 100
```

TYPBOUNDS: ОГРАНИЧЕНИЕ НА ТИП-АРГУМЕНТИ

```
class PairNumber <T extends Number> {
  private T first;
  private T second;

  PairNumber(T fst, T scd) {
     first = fst;
     second = scd;
  }

  public T getFirst() {
    return first;
  }

  public T getSecond() {
    return second;
  }

  public double add () {
    return first.doubleValue() + second.doubleValue();
  }
}
```

Тип-аргументът трябва да бъде изведен от Number:
Integer, Double …
(Wrapper-класове, които представят числа)

TYPBOUNDS: ОГРАНИЧЕНИЕ НА ТИП-АРГУМЕНТИ

```
class BuildPairsBounds {

  public static void main (String[] args) {
    PairNumber<Integer> pi;
//  PairNumber<String>  ps;
    Integer i, j;

    i = new Integer(99);
    j = new Integer(100);
    pi = new PairNumber<Integer> (i, j);

    System.out.println(pi.getFirst().intValue()
            + " " + pi.getSecond().intValue() + " "
            + pi.add());
  }
}
```

Integer изведен от Number

грешка!

```
% java BuildPairsBounds

99 100 199.0
```

БЛАГОДАРЯ ЗА ВНИМАНИЕТО!

КРАЙ “ГЕНЕТИЧНИ КЛАСОВЕ”

Distributed @learning center