NGÔN NGỮ LẬP TRÌNH JAVA

Nội dung

Cơ bản về ngôn ngữ lập trình Java

- Lập trình hướng đối tượng
- Biến, từ khoá, kiểu dữ liệu
- Biểu thức, các cấu trúc điều khiển
- Dữ liệu kiểu mảng

Các khía cạnh nâng cao của lập trình hướng đối tượng

- Thiết kế lớp
- Thiết kế lớp nâng cao
- Xử lý ngoại lệ

Xây dựng ứng dụng Java

- Tạo giao diện đồ hoạ
- Xử lý các sự kiện trên giao diện đồ hoạ
- Xây dựng ứng dụng đồ hoạ

Lập trình Java nâng cao

- Luồng
- Vào / Ra
- Lập trình mạng
- Lập trình với CSDL

BIỂU THỨC VÀ CÁC CẤU TRÚC ĐIỀU KHIỂN

Nội dung

- Phân biệt biến cục bộ, biến thể hiện
- Biến thể hiện được khởi tạo như thế nào?
- Sử dụng các phép toán trong Java
- Các biểu thức logic
- Sử dụng các cấu trúc điều khiển if, switch, for, while, do
- Sử dụng lệnh nhảy break, continue.

Biến và phạm vi hoạt động

- Biến cục bộ là:
  - Biến được định nghĩa bên trong một phương thức, người ta gọi nó là local, automatic, temporary, hay stack variable
  - Biến được tạo ra khi phương thức được thực thi và xoá bỏ khi phương thức kết thúc
- Biến cục bộ yêu cầu sự khởi tạo tường minh
- Biến thể hiện được khởi tạo 1 cách tự động

Biến và phạm vi hoạt động

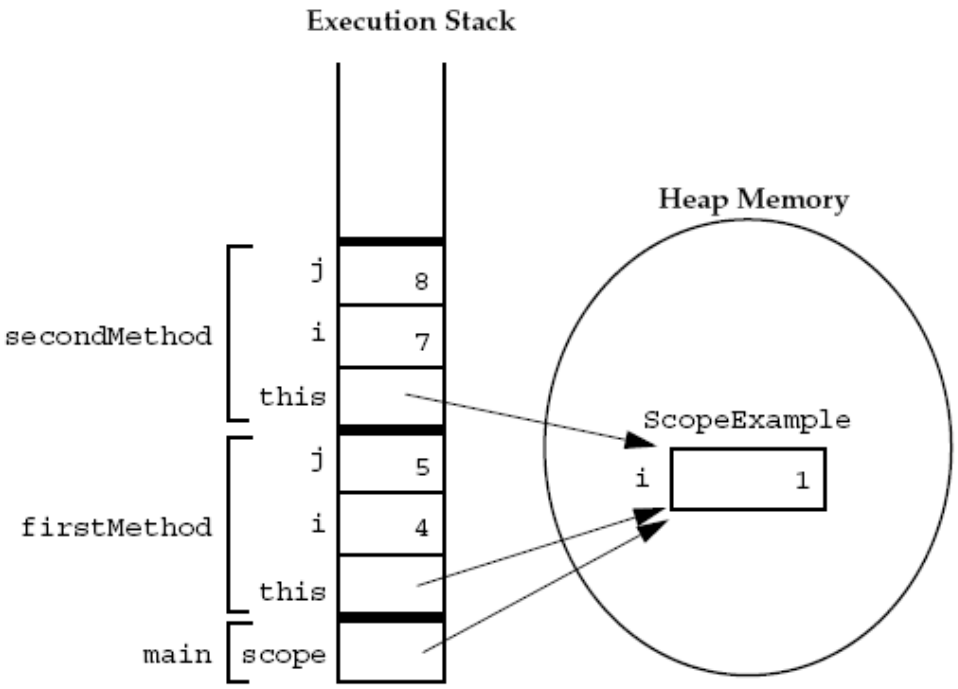
```
public class ScopeExample {
  private int i=1;

  public void firstMethod() {
    int i=4, j=5;

    this.i = i + j;
    secondMethod(7);
  }
  public void secondMethod(int i) {
    int j=8;
    this.i = i + j;
  }
}


public class TestScoping {
  public static void main(String[] args) {
    ScopeExample scope = new ScopeExample();

    scope.firstMethod();
  }
}
```



Giá trị mặc định của các kiểu dữ liệu

Variable	Value
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
char	'\u0000'
boolean	false
All reference types	null

Các phép toán và thứ tự ưu tiên

Operators	Associative
++ -- + *unary* - *unary* ~ ! (<*data_type*>)	R to L
* / %	L to R
+ -	L to R
<< >> >>>	L to R
< > <= >= instanceof	L to R
== !=	L to R
&	L to R
^	L to R
|	L to R
&&	L to R
||	L to R
<*boolean_expr*> ? <*expr1*> : <*expr2*>	R to L
= *= /= %= += -= <<= >>= >>>= &= ^= |=	R to L

Các phép toán logic

- The `boolean` operators are:

```
! - NOT     & - AND
| - OR      ^ - XOR
```

- The short-circuit `boolean` operators are:

```
&& - AND    || - OR
```

- You can use these operators as follows:

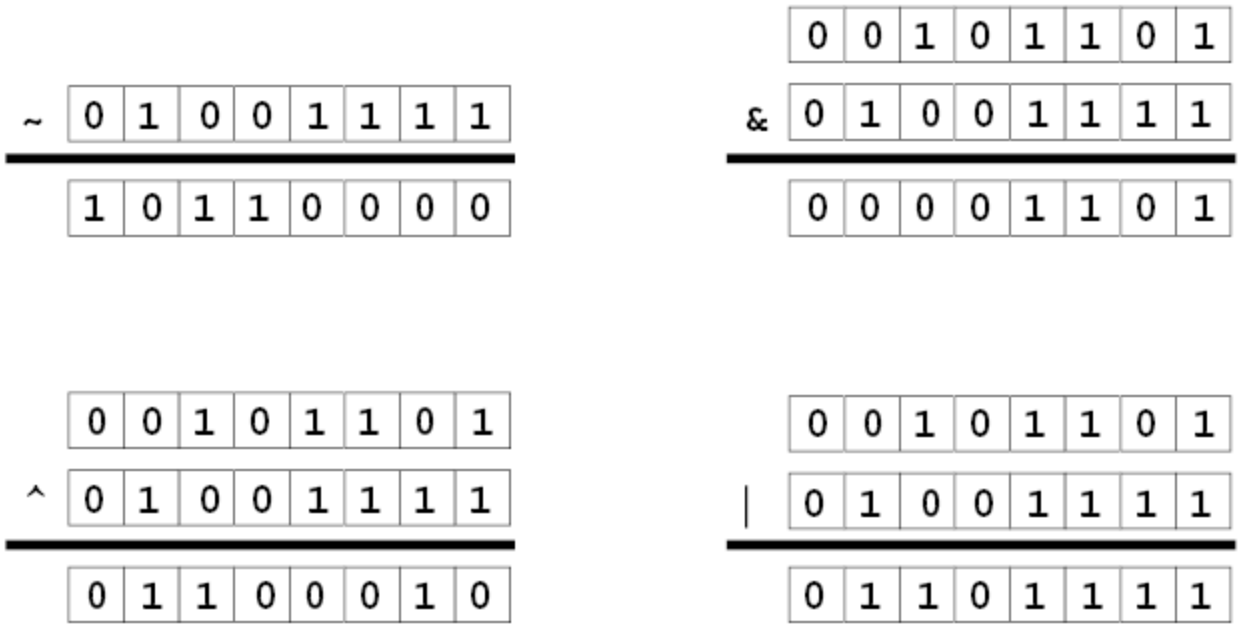
```
MyDate d = reservation.getDepartureDate();
if ( (d != null) && (d.day > 31) {
  // do something with d
}
```

Các phép toán trên bit

- The integer *bitwise* operators are:

```
~ - Complement   & - AND
^ - XOR          | - OR
```

- Byte-sized examples include:



Các phép dịch bit

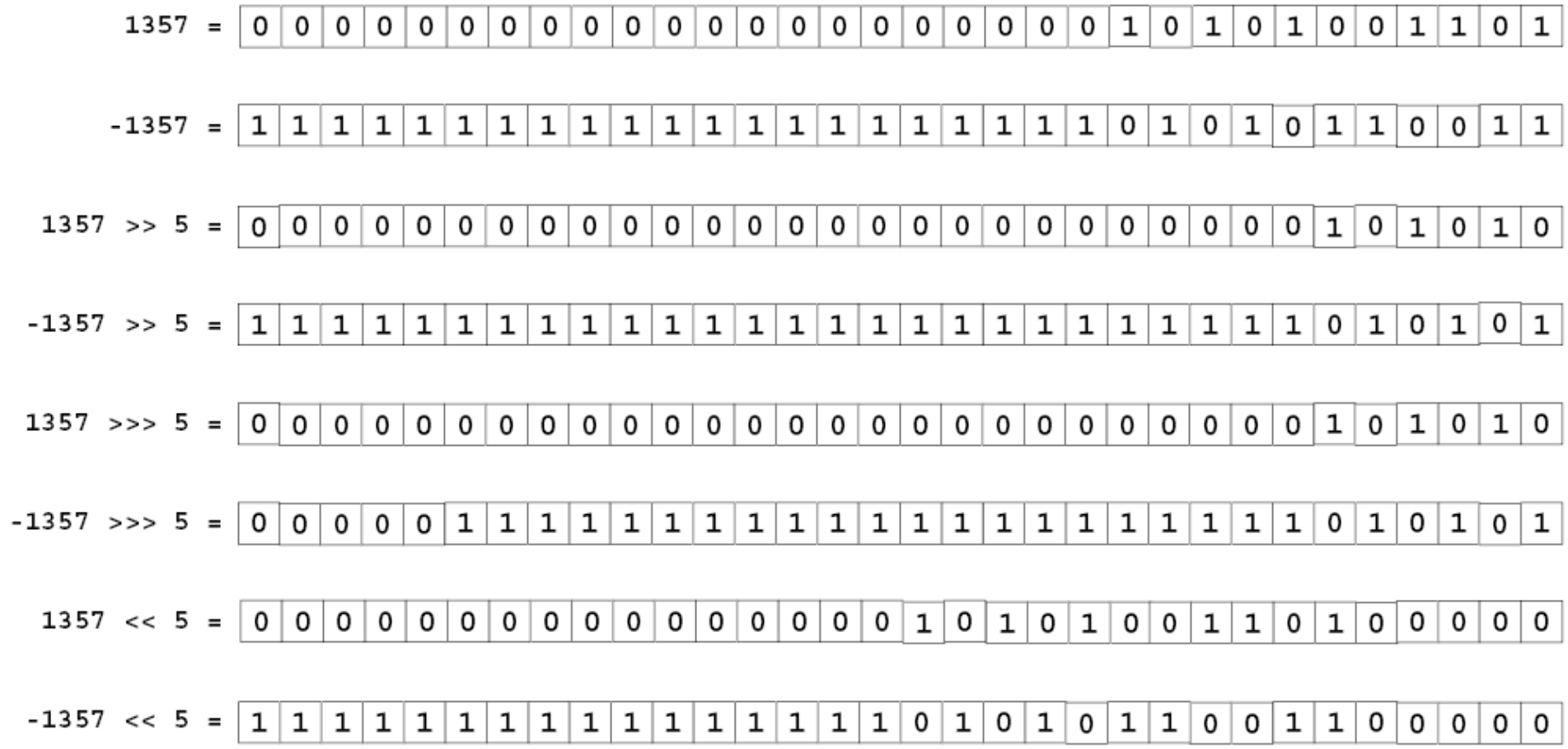
- 2 phép dịch phải:
  - dịch phải có cờ (signed right shift) : >>
  - dịch phải không cờ (unsigned right shift) : >>>

```
128 >> 1  returns  128/2^1  =  64
256 >> 4  returns  256/2^4  =  16
-256 >> 4 returns -256/2^4  = -16
```

- phép dịch trái: <<

```
128 << 1 returns 128 * 2^1 = 256
16  << 2 returns  16 * 2^2 = 64
```

Ví dụ về các phép dịch bit



Ép kiểu và nới kiểu

```
long bigValue = 99L;
int squashed = bigValue;        // Wrong, needs a cast
int squashed = (int) bigValue; // OK

int squashed = 99L;             // Wrong, needs a cast
int squashed = (int) 99L;       // OK, but...
int squashed = 99;              // default integer literal

long bigval = 6;      // 6 is an int type, OK
int smallval = 99L;   // 99L is a long, illegal

double z = 12.414F;   // 12.414F is float, OK
float z1 = 12.414;    // 12.414 is double, illegal
```

Câu lệnh if, else

- Cú pháp:

if (<boolean_expression>)
<statement_or_block>

```
if ( x < 10 ) {
    System.out.println("Are you finished yet?");
}
```

- Cú pháp:

if (<boolean_expression>)
<statement_or_block>
else
<statement_or_block>

```
if ( x < 10 ) {
System.out.println("Are you finished yet?");
} else {
System.out.println("Keep working...");
}
```

Câu lệnh switch

- Cú pháp:

```
switch ( <expression> ) {
  case <constant1>:
    <statement_or_block>*
    [break;]
  case <constant2>:
    <statement_or_block>*
    [break;]
  default:
    <statement_or_block>*
    [break;]
}
```

- Trong đó: `<expression>` phải trả về giá trị byte, short, int hoặc char
- Vị trí của default không quan trọng

Vòng lặp for

- Cú pháp:

```
for ( <A>; <B>; <D> )
{    <C>                }
```

- Trong đó:
  - **<A>:** biểu thức khởi tạo, chỉ chạy duy nhất 1 lần
  - **<B>:** điều kiện của vòng lòng, trả về true hoặc false
  - **<C>:** thân vòng lặp
  - **<D>:** cập nhật cho bước lặp

Vòng lặp while

- Cú pháp:

```
while ( <test_expr> )
    <statement_or_block>
```

- Ví dụ:

```
int i = 0;
while ( i < 10 ) {
    System.out.println(i + " squared is " + (i*i));
    i++;
}
```

Vòng lặp do - while

- Cú pháp:

```
do <statement_or_block>
while( <test_expr> );
```

- Ví dụ:

```
int i = 0;
do {
   System.out.println(i + " squared is " + (i*i));
   i++;
} while ( i < 10 );
```

Các lệnh nhảy

- Cú pháp:

break ***[<label>]***;

continue ***[<label>]***;

<label> : ***<statement>***

- Ví dụ:

```
1 do {
2 statement;
3 if ( condition ) {
4     break;
5 }
6 statement;
7 } while ( test_expr );
```

```
1 do {
2 statement;
3 if ( condition ) {
4     continue;
5 }
6 statement;
7 } while ( test_expr );
```

Các lệnh nhảy

```
1 outer:
2 do {
3   statement1;
4   do {
5     statement2;
6     if ( condition ) {
7       break outer;
8     }
9     statement3;
10  } while ( test_expr );
11  statement4;
12 } while ( test_expr );
```

```
1 test:
2 do {
3   statement1;
4   do {
5     statement2;
6     if ( condition ) {
7       continue test;
8     }
9     statement3;
10  } while ( test_expr );
11  statement4;
12 } while ( test_expr );
```

KIỂU DỮ LIỆU MẢNG

Nội dung

- Khai báo và khởi tạo mạng có kiểu dữ liệu gốc
- Giải thích tại sao mảng là kiểu dữ liệu đối tượng.
- Khởi tạo giá trị phần tử mảng
- Khai báo và khởi tạo mảng nhiều chiều
- Sao chép các phần từ mảng

Định nghĩa - khai báo mảng

- Mảng là một tập hợp các đối tượng dữ liệu có cùng kiểu
- Ví dụ về khai báo mảng:

```
char s[];
Point p[];
char[] s;
Point[] p;
```

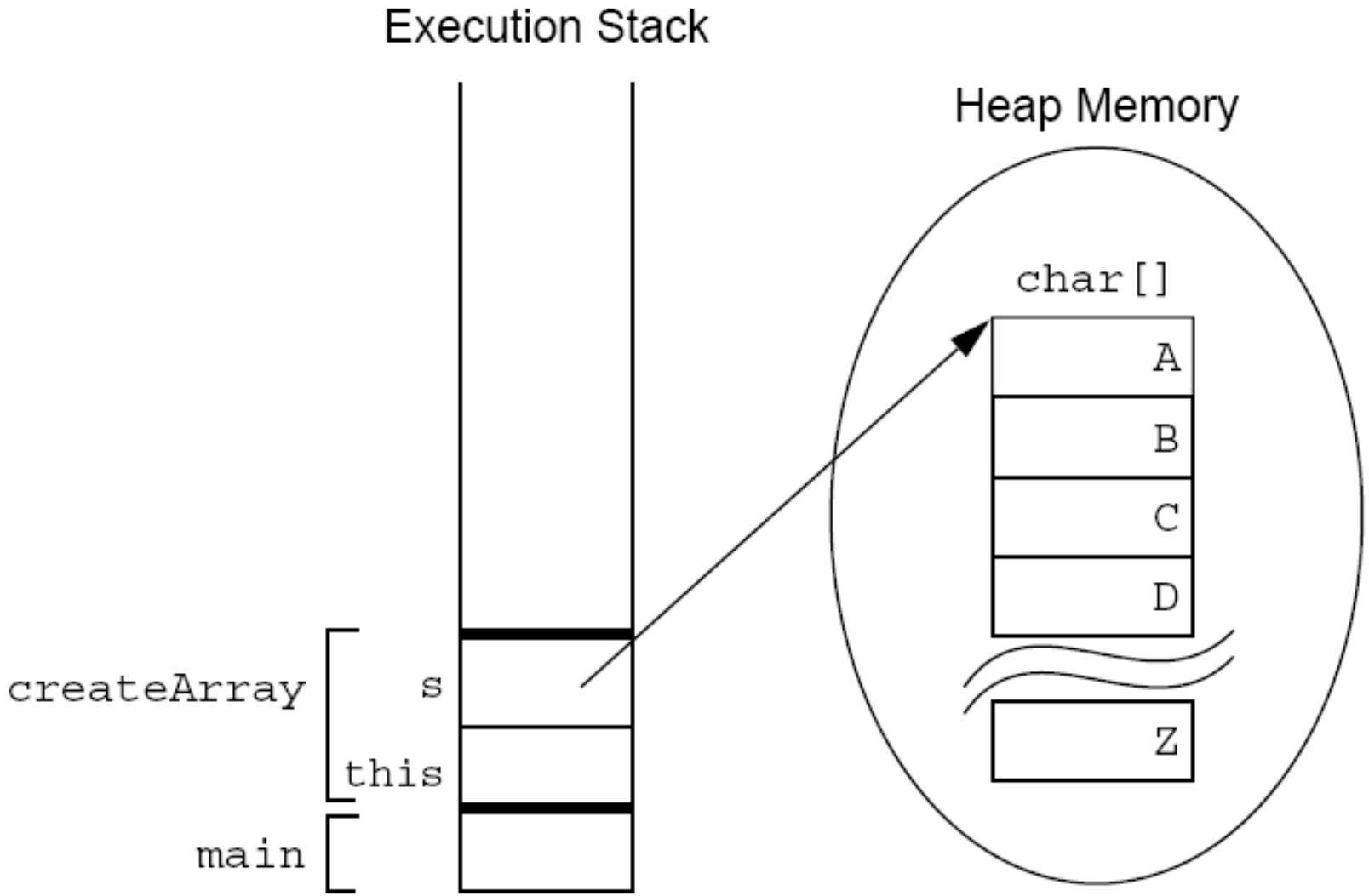
- Mảng là đối tượng. Nó được khởi tạo bởi từ khoá new

Khởi tạo mảng

- Sử dụng từ khoá new để tạo mới mảng
- Ví dụ về khởi tạo mảng:

```
1 public char[] createArray() {
2     char[] s;
3
4     s = new char[26];
5     for ( int i=0; i<26; i++ ) {
6         s[i] = (char) ('A' + i);
7     }
8
9     return s;
10 }
```

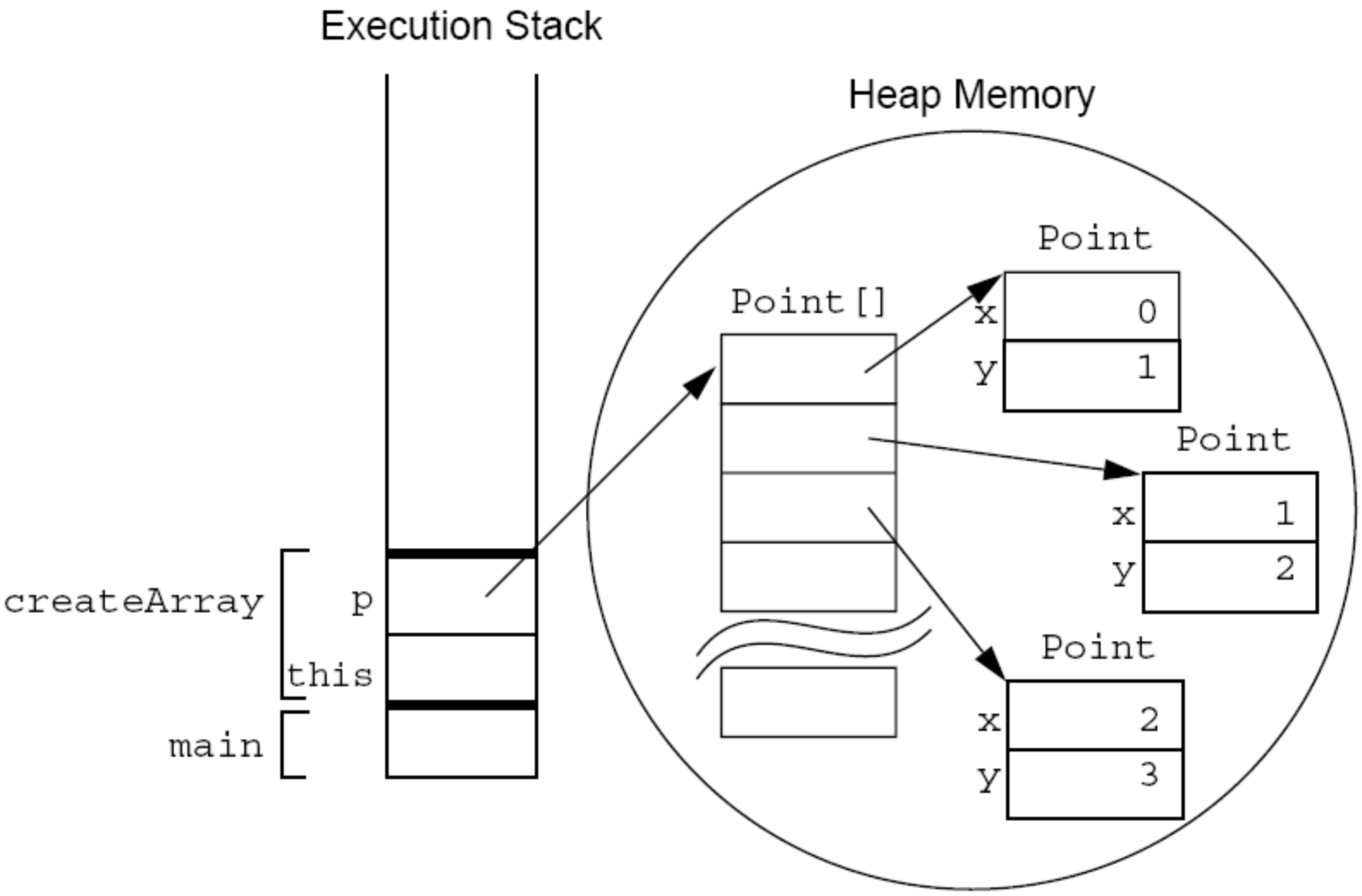
Lưu trữ dữ liệu trong mảng



Tạo mảng các đối tượng

```
1 public Point[] createArray() {
2     Point[] p;
3
4     p = new Point[10];
5     for ( int i=0; i<10; i++ ) {
6         p[i] = new Point(i, i+1);
7     }
8
9     return p;
10 }
```

Lưu trữ dữ liệu phân cấp trong mảng



Khởi tạo các phần tử trong mảng

```
String[] names;
names = new String[3];
names[0] = "Georgianna";
names[1] = "Jen";
names[2] = "Simon";
```

```
String[] names = {
    "Georgianna",
    "Jen",
    "Simon"
};
```

```
MyDate[] dates;
dates = new MyDate[3];
dates[0] = new MyDate(22, 7, 1964);
dates[1] = new MyDate(1, 1, 2000);
dates[2] = new MyDate(22, 12, 1964);
```

```
MyDate[] dates = {
    new MyDate(22, 7, 1964),
    new MyDate(1, 1, 2000),
    new MyDate(22, 12, 1964)
};
```

Mảng nhiều chiều

```
int[][] twoDim = new int[4][];
twoDim[0] = new int[5];
twoDim[1] = new int[5];

int[][] twoDim = new int[][4]; // illegal
```

- Duyệt mảng bắt đầu từ chỉ số 0

```
public void printElements(int[] list) {
  for (int i = 0; i < list.length; i++) {
    System.out.println(list[i]);
  }
}
```

- Vòng for cải tiến để duyệt mảng:

```
public void printElements(int[] list) {
  for ( int element : list ) {
    System.out.println(element);
  }
}
```

Ghi nhớ về mảng

- Mảng không thể thay đổi kích thước. Muốn thay đổi phải tạo mảng mới.
- Có thể dùng cùng một biến tham chiếu để trỏ tới một mảng mới.

```
int[] myArray = new int[6];
myArray = new int[10];
```

Sao chép các phần tử mảng

- Sử dụng phương thức System.arraycopy() để sao chép phần tử mảng

```
1   //original array
2   int[] myArray = { 1, 2, 3, 4, 5, 6 };
3
4   // new larger array
5   int[] hold = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
6
7   // copy all of the myArray array to the hold
8   // array, starting with the 0th index
9   System.arraycopy(myArray, 0, hold, 0, myArray.length);
```