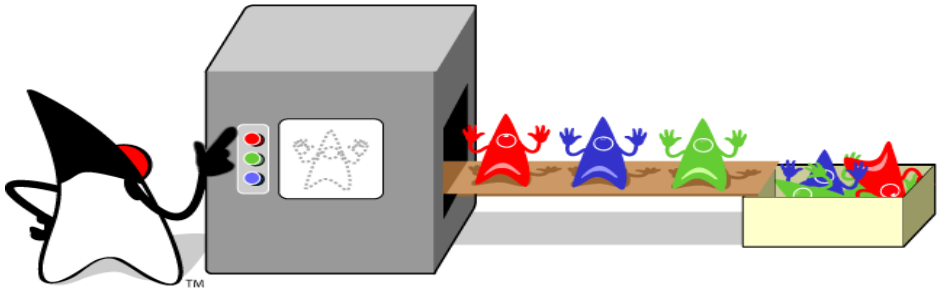
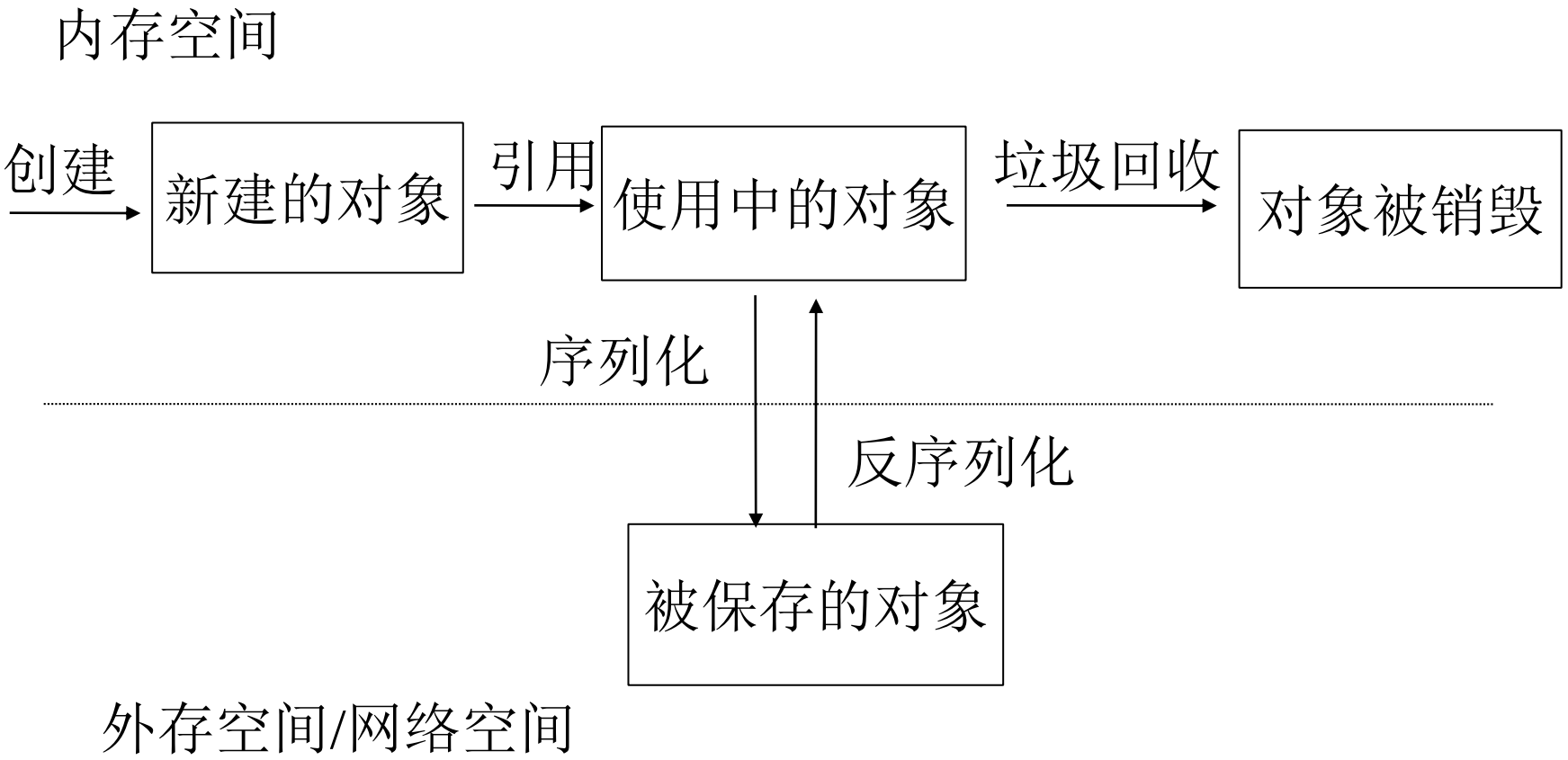
Chap 4 Java对象生命周期和常用对象



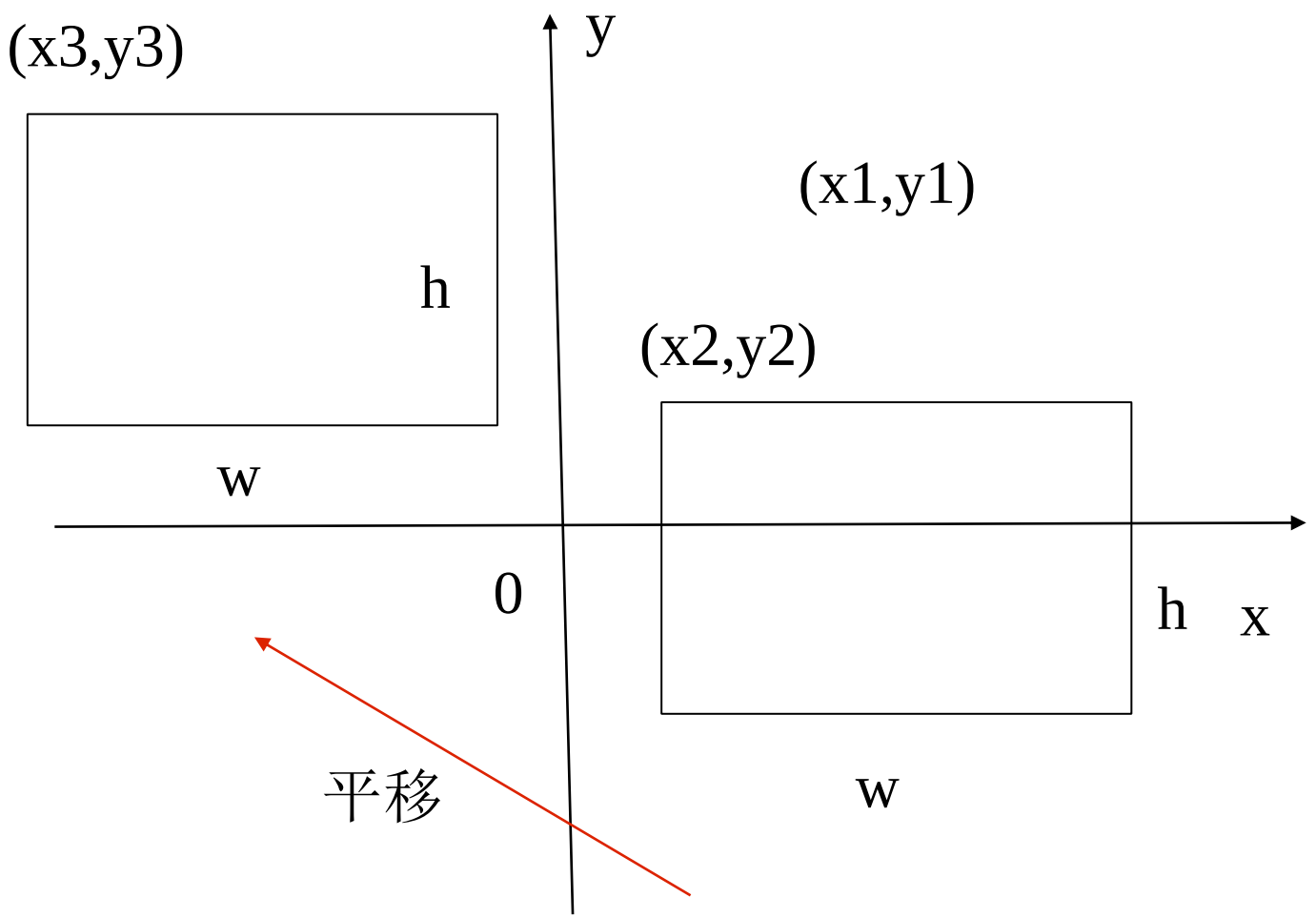
4.1 Java虚拟机中的对象的创建，使用和垃圾回收

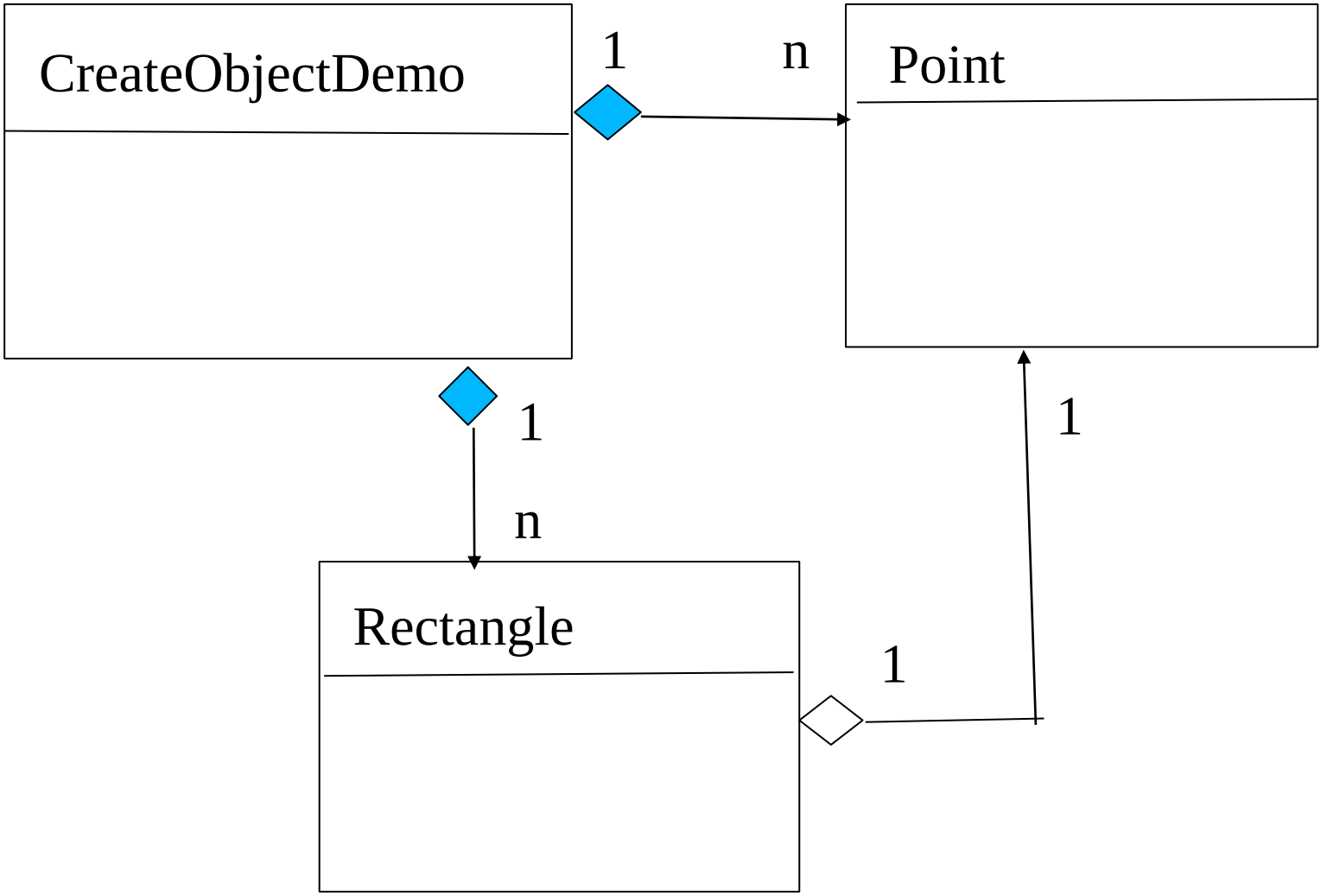
对象的生命周期

通常情况下，一个Java程序中将会创建很多对象。当这些对象完成了相应工作任务后，将被进行垃圾回收(garbage-collected) ，所占用的资源回收后重新分配给其他对象使用。



Java 对象生命周期


(x3,y3)
y
(x1,y1)
h
(x2,y2)
w
0
h
x
平移
w


CreateObjectDemo
1
n
Point
1
n
1
Rectangle
1

举例:

```
public class CreateObjectDemo {
    public static void main(String[] args) {

        // 创建1个 point 对象和两个rectangle对象
        Point origin_one = new Point(23, 94);
        Rectangle rect_one = new Rectangle(origin_one, 100, 200);
        Rectangle rect_two = new Rectangle(50, 100);
        //显示rect_one 的width, height, and area
        System.out.println("Width of rect_one: " + rect_one.width);
        System.out.println("Height of rect_one: " + rect_one.height);
        System.out.println("Area of rect_one: " + rect_one.area());
        // 设置rect_two 的位置
        rect_two.origin = origin_one;
```

```
        //显示rect_two的位置
        System.out.println("X Position of rect_two: " + rect_two.origin.x);
        System.out.println("Y Position of rect_two: " + rect_two.origin.y);

        //移动rect_two并显示其新的位置
        rect_two.move(40, 72);
        System.out.println("X Position of rect_two: " + rect_two.origin.x);
        System.out.println("Y Position of rect_two: " + rect_two.origin.y);
    }
}
```

4.1.1 创建对象

Point 类的代码:

```
public class Point {
    public int x = 0;
    public int y = 0;
    //构造方法
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Point origin_one = new Point(23, 94);
Rectangle rect_one = new Rectangle(origin_one, 100, 200);
Rectangle rect_two = new Rectangle(50, 100);

语句包含三个部分:
声明；实例化；初始化

- 声明一个引用对象的变量

变量类型为类或接口的名称，在声明时对象并没有被创建。

Point origin_one = new Point(23, 94);

- 初始化和实例化一个对象

使用**new**操作符实例化一个类并为得到的对象分配内存空间。

Point origin_one = new Point(23, 94);

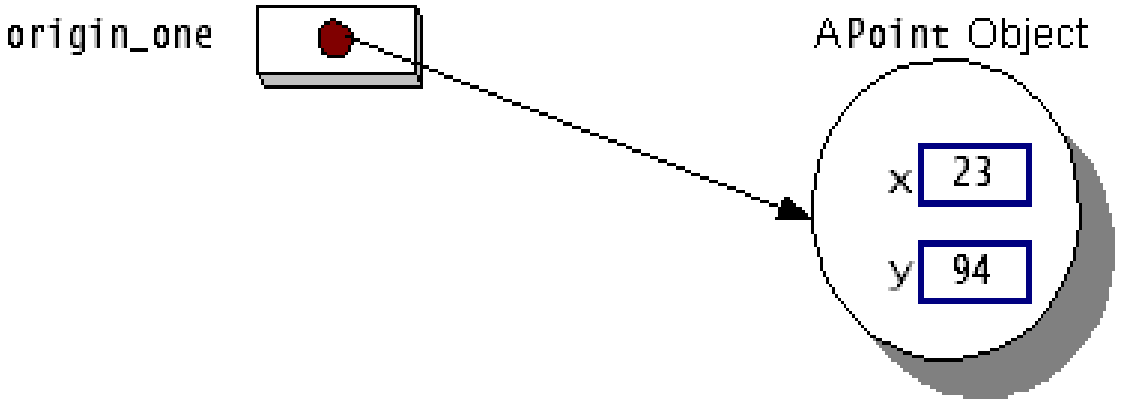
类的构造方法名称作为**new**操作符的参数，构造方法的作用是负责对新得到的对象进行初始化。

Point origin_one = new Point(23, 94);

new操作符最后返回对创建完成的对象的一个引用。这个引用通过赋值操作传递给预先定义的引用变量。

Point origin_one = new Point(23, 94);

Point origin_one = new Point(23, 94);



Rectangle类的代码

```
public class Rectangle {
    public int width = 0;
    public int height = 0;
    public Point origin;
    //4个构造方法
    public Rectangle() {
        origin = new Point(0, 0);
    }

    public Rectangle(Point p) {
        origin = p;
    }
```

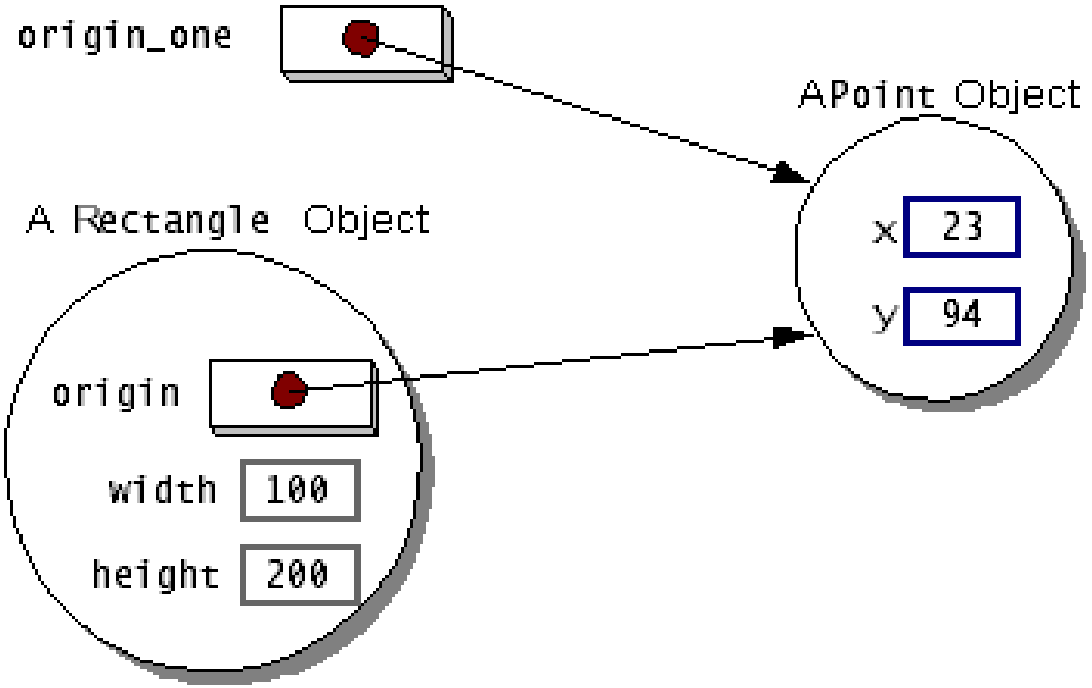
```
public Rectangle(int w, int h) {
    this(new Point(0, 0), w, h);
}
public Rectangle(Point p, int w, int h) {
    origin = p;
    width = w;
    height = h;
}

//移动rectangle的方法
public void move(int x, int y) {
    origin.x = x;
    origin.y = y;
}
```

```
   //计算rectangle的面积
    public int area() {
        return width * height;
    }
}
```

```
Rectangle rect = new Rectangle();   //无参数构造方法
Rectangle rect_two = new Rectangle(50, 100);
Rectangle rect_one = new Rectangle(origin_one, 100, 200);
```

注意：如果一个类没有明确地定义其构造方法，Java平台会自动为其提供一个无参数的构造方法，也称为默认的构造方法，但这个构造方法不完成任何实际的工作。因此所有的类都至少会有一个构造方法。

Rectangle rect_one = new Rectangle(origin_one, 100, 200);



4.1.2 使用对象

对象被创建以后，可以对其进行操作，使用其变量或调用其方法。

• 对对象变量的引用
限制命名的一般形式: 对象引用名.变量名

注意! 简单命名和限定命名
举例: 在**Rectangle** 对象中

```
width = w;
System.out.println("Width of rect_one: " + rect_one.width);
```

注意! 不建议在对象外部直接对一个对象的变量进行操作，而是使用对象的方法对其变量进行操作。

```
public class Rectangle {
    public int width = 0;
    public int height = 0;
    public Point origin;
     ……
//使用setXXX 方法设置实例变量的值
    public setWidth(int w) {
        width = w;
    }

//使用getXXX 方法获取实例变量的值
   public int getWidth(){
     return width;
   }
      ……
}
```

```
int width_value;

Rectangle rectOne = new Rectangle(originOne, 100, 200);

width_value = rectOne.getWidth(); //建议方式

width_value = rectOne.width;  //不建议方式
```

- 对对象方法的调用

对象引用名.方法名( );
或
对象引用名.方法名(参数表);

举例:

```
System.out.println("Area of rect_one: " + rect_one.area());

rect_two.move(40, 72);
```

注意！对对象成员的访问控制机制，将在第5章讲解。

4.1.3 无用对象和垃圾回收算法

Java运行环境检查到某个对象不再被使用时，将会把它删除掉，这一过程称为垃圾回收(garbage collection)

一个对象被垃圾回收的条件（无用对象）是：对这个对象的引用数为0，或在遍历对象树不可达。

垃圾回收器(Garbage Collector)

Java运行环境使用垃圾回收器周期性地释放掉那些被回收的无用对象占用的内存空间。垃圾回收器的工作是自动进行的。

举例：

```
Class Myclass{

public static void main(String args[]){

  Object  a = new Object();
  Object  b = new Object();
  Object  c = new Object();
  b = a;
  c = a;
  a = null;
  b = null;
  c = null;
  }
}
```

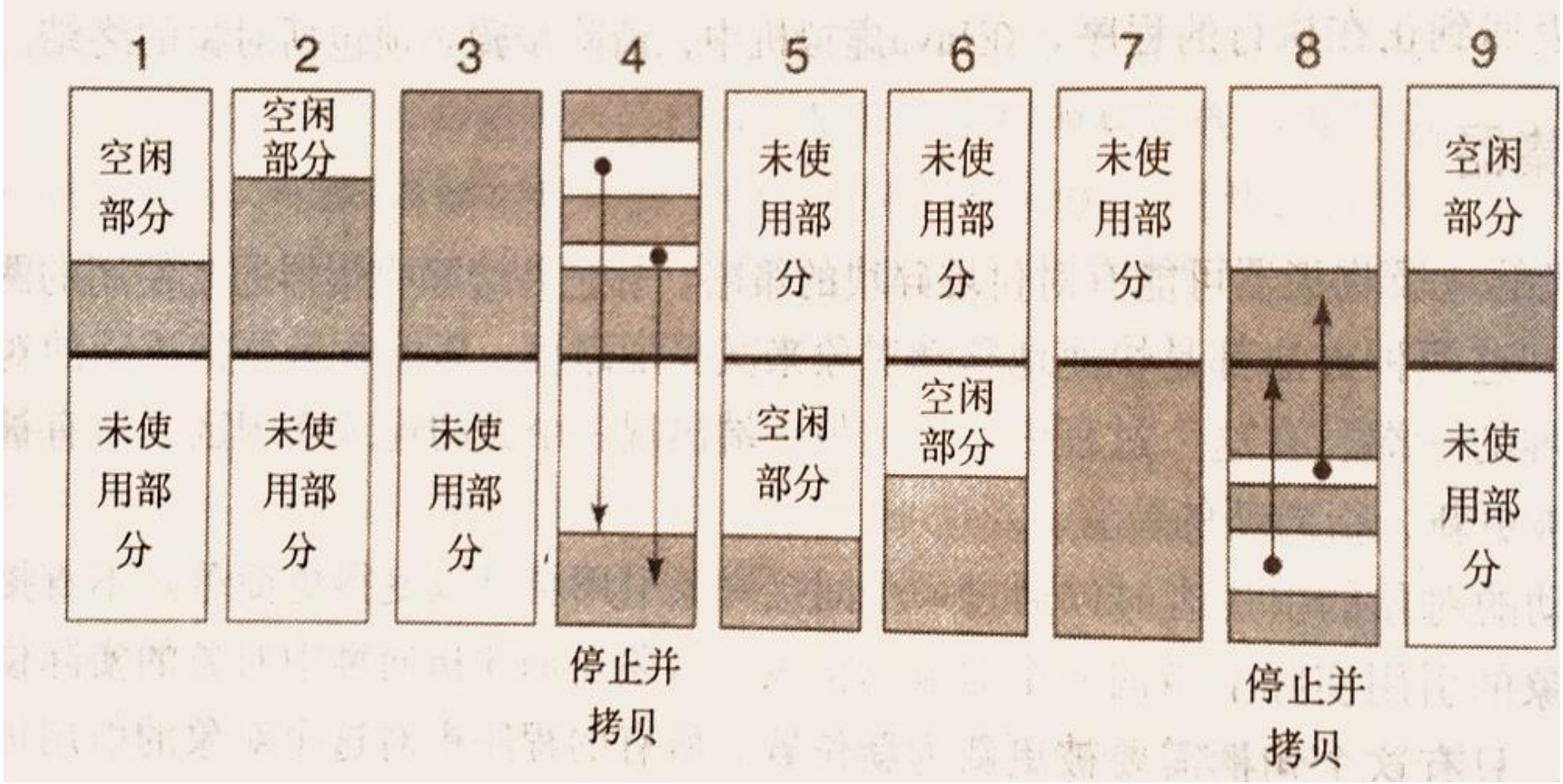
经典垃圾回收算法

回收算法:

1. 检测出无用对象（引用计数器，对象树遍历）

2. 回收无用对象占用的存储空间，整理存储空间减少碎片
（停止拷贝算法，标记清扫算法）

停止拷贝算法(stop & copy)



一个堆中的停止拷贝算法执行过程

简单的停止拷贝算法在每次拷贝时，所有的活动对象均需要被拷贝。在对象生命周期很长时，效率较低。

改进后的停止拷贝算法：将对象按照寿命（代）分组，回收年幼的无用对象。将堆划分为子堆，把年长的对象放入代数大的子堆，降低回收的频度。

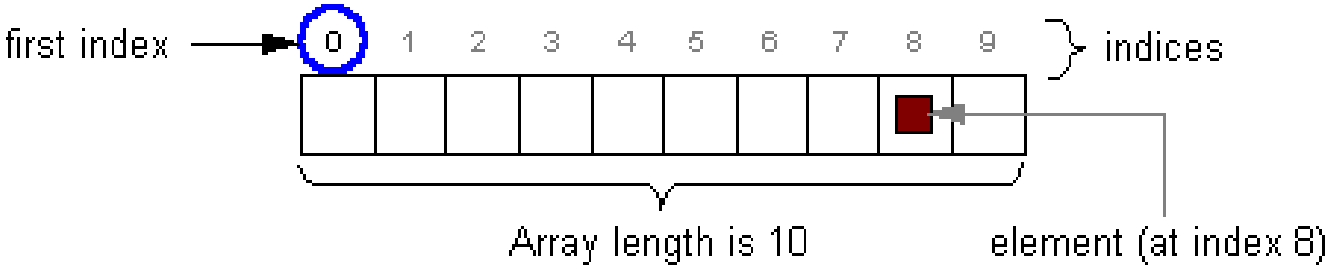
成熟的按代垃圾回收算法：火车算法(Richard Hudson,Eliot Moss),目前用于Java hotspot虚拟机。

标记清扫算法(mark & sweep)

标记清扫算法配合对象遍历操作，标记不可达的无用对象，回收其占用的存储资源。同时，把活动对象推过空闲区，滑动到堆的另一端，即进行堆活动空间的压缩过程。

4.2 数组对象(Arrays)

数组是可以存储多个相同类型数据的固定长度存储结构.



注意! 可以使用Collection(比如Vector 结构) 来存储不同类型数据

4.2.1 创建和使用数组

举例:

```
public class ArrayDemo {
   public static void main(String[] args) {
      int[] anArray;            // declare an array of integers
      anArray = new int[10];        // create an array of integers
      // assign a value to each array element and print
      for (int i = 0; i < anArray.length; i++) {
         anArray[i] = i;
         System.out.print(anArray[i] + " ");
      }
      System.out.println();
   }
}
```

- 声明一个变量来引用数组

int[] anArray;
float[] anArrayOfFloats;
String[] anArrayOfStrings;

注意! 声明仅仅创建数组引用，但不创建数组对象

- 创建数组对象

可以使用 **new** 操作符创建数组

new elementType[arraySize]

anArray = new int[10];

- 初始化和访问数组元素

anArray[i] = i;

- 获取数组长度

anArray.length

4.2.2对象的数组

数组可以存放基本类型数据也可存放对象引用

举例:

```
public class ArrayOfIntegersDemo {
    public static void main(String[] args) {
        Integer[] anArray = new Integer[10];
        for (int i = 0; i < anArray.length; i++) {
            anArray[i] = new Integer(i);
            System.out.println(anArray[i]);
        }
    }
}
```

注意！Integer[] anArray = new Integer[10];
仅仅创建含有10个指向Integer引用的数组，但每一个Integer对象需要在之后逐个进行创建

4.2.3 数组的数组

数组可以包含数组，构成多维数组

举例: 矩阵

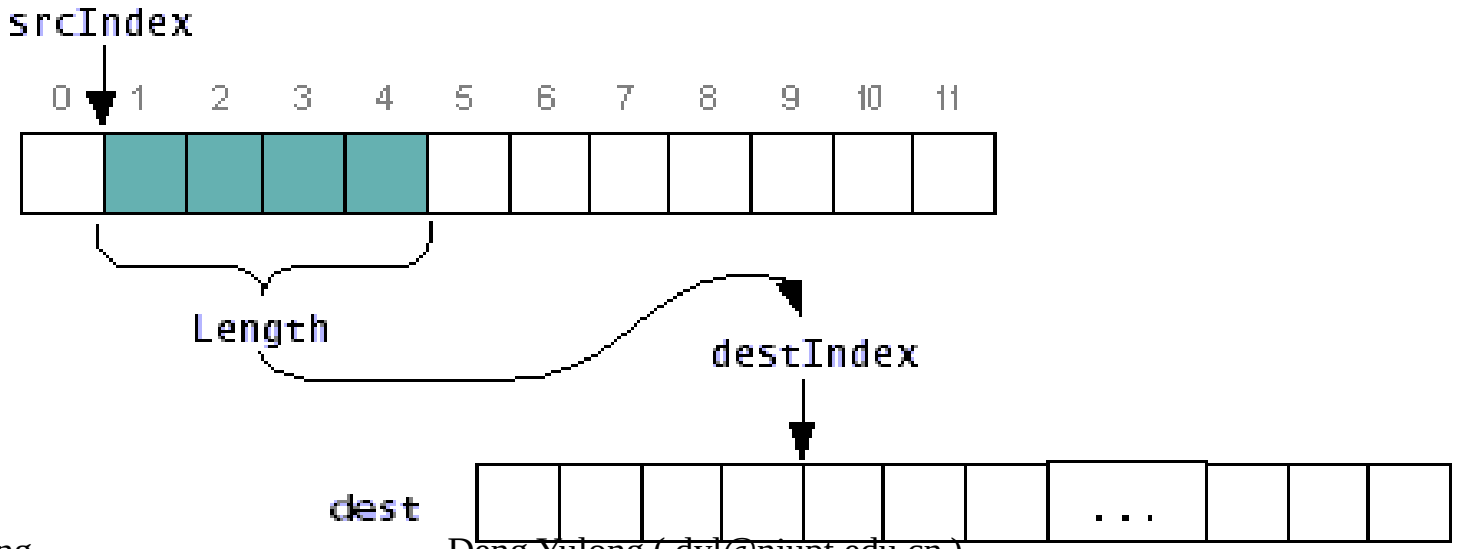
```
public class ArrayOfArraysDemo2 {
    public static void main(String[] args) {
        int[][] aMatrix = new int[4][];

         //populate matrix
        for (int i = 0; i < aMatrix.length; i++) {
            aMatrix[i] = new int[5];   //create sub-array
            for (int j = 0; j < aMatrix[i].length; j++) {
                aMatrix[i][j] = i + j;
            }
        }
```

```
    //print matrix
      for (int i = 0; i < aMatrix.length; i++) {
          for (int j = 0; j < aMatrix[i].length; j++) {
             System.out.print(aMatrix[i][j] + " ");
          }
          System.out.println();
      }
   }
}
```

4.2.4 数组复制

使用**arraycopy** 方法来从一个数组复制数据到另一个数组

public static void arraycopy(Object source, int srcIndex,Object dest, int destIndex, int length)



举例:

```
public class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                            'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

4.3 字符对象(Character)和字符串对象(String)

定义：

* 字符是一个类，它的实例可以存放一个单个的字符型数值。同时，字符类也定义了对单个字符数据的一些操作。

* 字符串是一个类，它用于存储和操作由若干个字符数值构成的不可变数据结构。

4.3.1 字符

字符对象和基本型字符变量

使用单个字符值，采用基本字符变量 **char**
使用字符类的功能，则需要封装成为字符类对象
字符类同时提供了与字符处理相关的有用功能

举例:

```
Character a = new Character('a');
Character a2 = new Character('a');
Character b = new Character('b');

a.compareTo(b);
a.toString();
Character.isUpperCase(a.charValue());
```

举例：

```
Character a = new Character('a');
if (Character.isUpperCase(a.charValue()){
   System.out.println("The character " + aChar + " is upper case.");
} else {
   System.out.println("The character " + aChar + " is lower case.");
}
```

4.3.2 字符串

字符串类可以实现字符串结构。

举例: (对一个字符串的字符进行反转操作)

```
public class StringsDemo {

    public static void main(String[] args) {

        String palindrome = "Dot saw I was Tod";
        int len = palindrome.length();
      char[] tempCharArray = new char[len];
      char[] charArray = new char[len];
```

```
    for (int i = 0; i < len; i++) {
          tempCharArray[i] = palindrome.charAt(i);
      }


      for (int j =0; j  < len ; j++) {
         charArray[j] = tempCharArray[len – 1– j];
      }

      String reversePalindrome = new String(charArray);

       System.out.println(reversePalindrome);
   }
}
```

4.3.3 创建字符串

字符串通常由通过双引号包括起来的一串字符进行创建。

```
String palindrome = "Dot saw I was Tod";

String palindrome = new String("Dot saw I was Tod");
```

4.3.4 得到字符串的长度

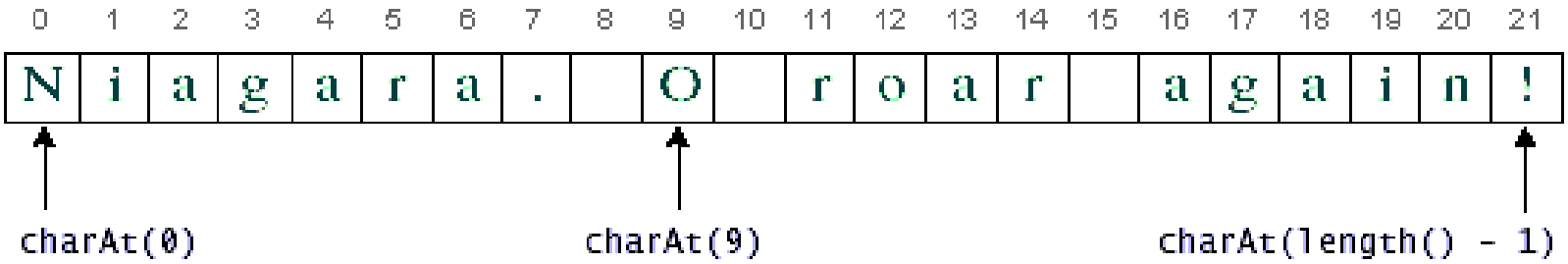
length 方法可以返回字符串中字符的个数，即字符串的长度。

```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();  // len = 17
```

4.3.5通过索引从字符串中取得对应字符

charAt 方法返回一个字符。

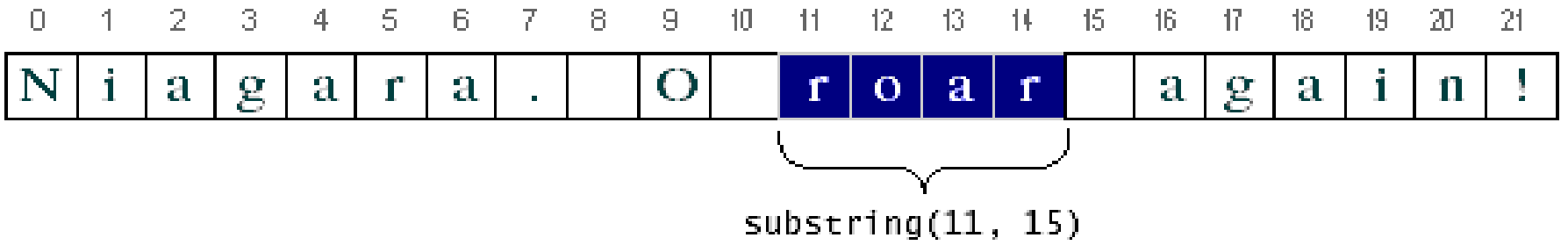
example
String anotherPalindrome = "Niagara. O roar again!";
char aChar = anotherPalindrome.charAt(9);



substring 方法返回一个以上的字符

example:

```
String anotherPalindrome = "Niagara. O roar again!";
String roar = anotherPalindrome.substring(11, 15);
```



4.3.6 在字符串内搜索字符或子串

indexOf 方法(从字符串开头进行 ->)
lastIndexOf 方法(<- 从字符串结尾进行)

4.3.7 字符串比较

endsWith, startsWith, compareTo, compareToIgnoreCase, equals, equalsIgnoreCase, regionMatches 方法

4.3.8 字符串操作

concat, replace, trim, toLowerCase, toUppercase 方法

4.3.9字符串和编译器

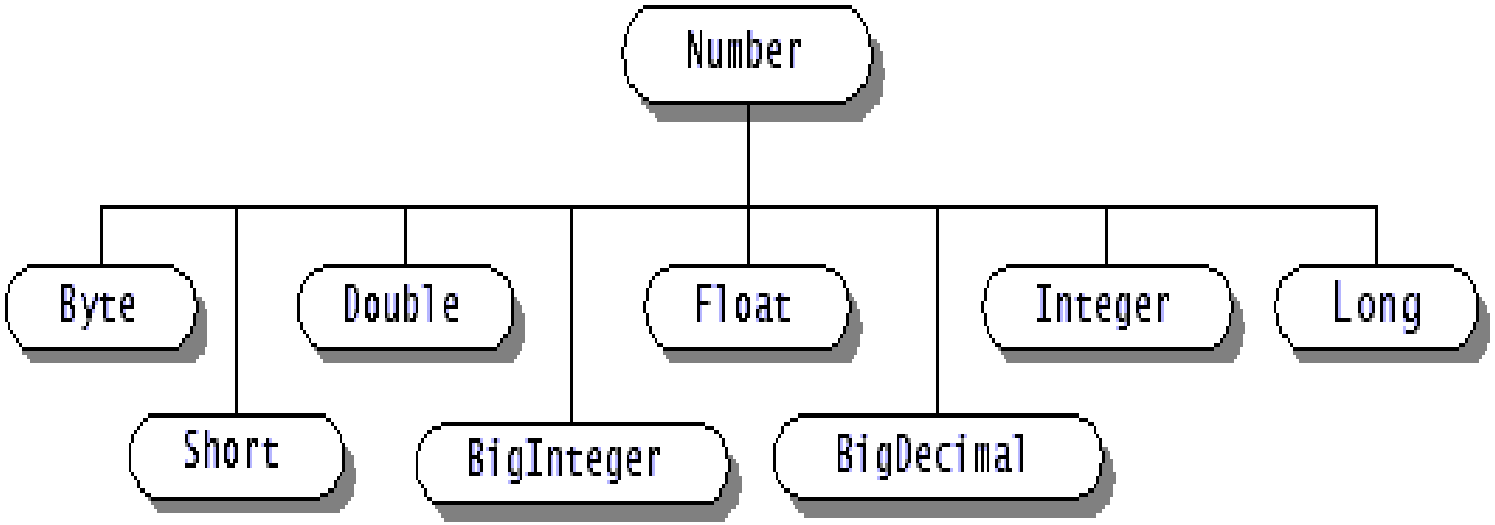
•在使用字符串对象的场合可以直接使用“字符”构成的字符串。
举例: System.out.println(“hello”);
•可以直接在“字符”构成的字符串基础上使用字符串方法。
举例: int len = “hello”.length() ;
• 可以使用 + 连接几个字符串。
举例: System.out.println("hello"+" everyone");
System.out.println("hello"+1000);

注意! String s =new String("hello"); and String s ="hello";

4.4 基本数据类型对象(数值类)(Numbers)

4.4.1 Number 类

Number类是所有的数值类(number)的父类



类型包装类包括: **number, Boolean, Character, Void.**

类型包装类

*** 类型包装类对象中可以存储一个基本类型变量数值**

*** 类型包装类定义了有用的变量如：MIN_VALUE and MAX_VALUE, 以及有用的方法如：,将数值转换为字符串**

```
byte largestByte = Byte.MAX_VALUE     (127)

---------

Integer intOne = new Integer(10);

String strOne = intOne.toString();
```

4.4.2将字符串转换为数字

获取用户输入 (String -> numbers)

举例:

```
public class ValueOfDemo {
    public static void main(String[] args) {

        //this program requires two arguments on the command line
        if (args.length == 2) {

            //convert strings to numbers
            float a = Float.valueOf(args[0]).floatValue();
            float b = Float.valueOf(args[1]).floatValue();
```

```
            //do some arithmetic
            System.out.println("a + b = " + (a + b) );
            System.out.println("a - b = " + (a - b) );
            System.out.println("a * b = " + (a * b) );
            System.out.println("a / b = " + (a / b) );
            System.out.println("a % b = " + (a % b) );
        } else {
            System.out.println("This program requires two command-line
             arguments.");
        }
    }
}
```

4.4.3 将数字转换为字符串

Numbers -> Strings.

举例:

```
public class ToStringDemo {
    public static void main(String[] args) {
        double d = 858.48;
        String s = Double.toString(d);
        int dot = s.indexOf('.');
        System.out.println(s.substring(0, dot).length()
                           + " digits before decimal point.");
        System.out.println(s.substring(dot+1).length()
                           + " digits after decimal point.");
    }
}
```

4.4.4 数学运算
基本数学运算 (java.lang)
高级数学运算 (java.lang.math)

abs, ceil, floor, rint, round, min, max, exp, log, pow, sqrt, sin, cos, tan, asin, acos, atan, atan2, toDegrees, toRadians, random

注意! 类方法: 举例. Math.log(x)
Math.PI
Random number: int number = (int)(Math.random()*10 + 1);