



Hiểu thêm về Java



Nội dung

- Dữ liệu kiểu nguyên thủy và đối tượng
- Tham chiếu
- Giải phóng bộ nhớ
- Gói và kiểm soát truy cập
- Kiểu hợp thành (composition)
- Vào ra với luồng dữ liệu chuẩn



Tài liệu tham khảo

- *Giáo trình Lập trình HĐT*, chương 2, 3, 4
- *Thinking in Java*, chapter 2, 4, 5
- *Java how to program*, chapter 4,5,6,7,8



Kiểu dữ liệu nguyên thủy

- Java cung cấp các kiểu nguyên thủy
 - số: byte, short, int, long, float, double
 - không có khái niệm unsigned
 - kích thước cố định trên mọi platform
 - logic: boolean
 - ký tự: char
- Dữ liệu kiểu nguyên thủy không phải là đối tượng
 - `int a = 5;`
 - `if (a==b)...`
- Tồn tại lớp đối tượng tương ứng: Integer, Float,..
 - `Integer count = new Integer(0);`



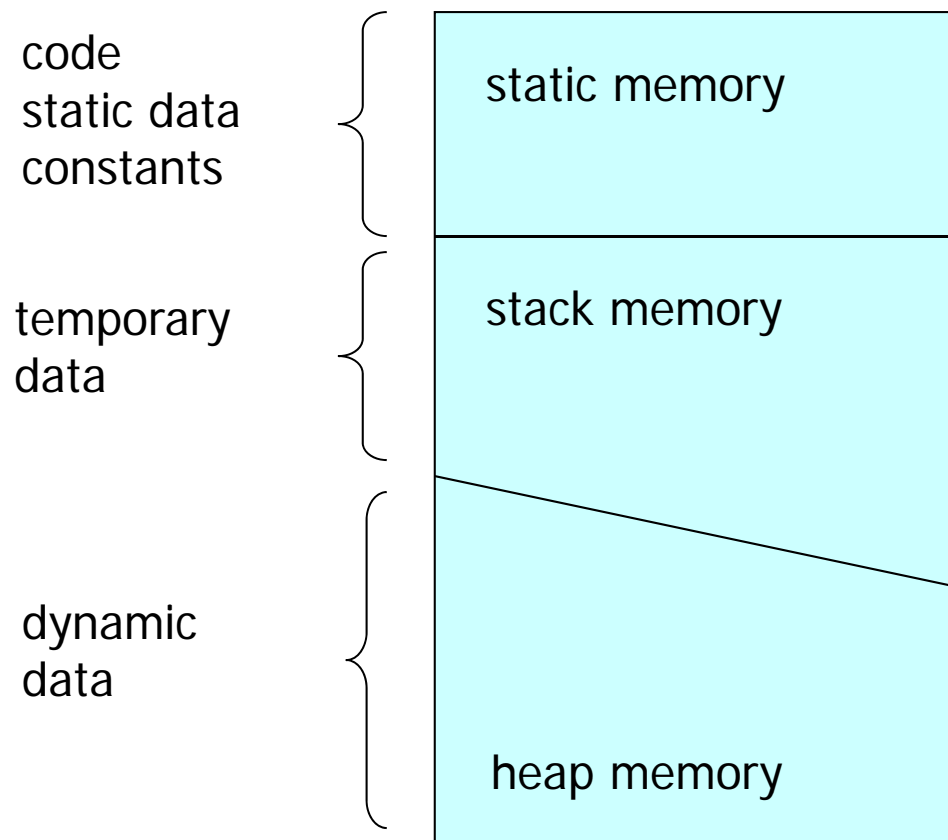
Kiểu dữ liệu	Độ rộng (bits)	Giá trị cực tiểu	Giá trị cực đại
char	16	0x0	0xffff
byte	8	-128 (-2^7)	+127 (2^7-1)
short	16	-32768 (-2^{15})	32767 ($2^{15}-1$)
int	32	-2^{31} , 0x80000000	$+2^{31} - 1$, 0x7fffffff
long	64	-2^{63}	$+2^{63} - 1$
float	32	1.40129846432481707e-45	3.40282346638528860e+38
double	64	4.94065645841246544e-324	1.79769313486231570e+308
boolean			




Dữ liệu được lưu trữ ở đâu

- Dữ liệu kiểu nguyên thủy
 - Thao tác thông qua *tên biến*
- Dữ liệu được đóng gói (là thuộc tính) trong đối tượng
 - Đối tượng được thao tác thông qua tham chiếu
- Vậy biến kiểu nguyên thủy, tham chiếu và đối tượng được lưu trữ ở đâu?

3 vùng bộ nhớ cho ứng dụng





```
class Countdown {
    Countdown(int n) {
        System.out.println(n);
        if (n>0) countdown(n);
    }
    int countdown(int n) {
        Countdown c;
        c = new Countdown(n-1);
        return n;
    }
}
...
new Countdown(10);
...
```




Tham chiếu

- Đối tượng được thao tác thông qua tham chiếu
 - là *con trỏ* tới đối tượng
 - thao tác trực tiếp tới thuộc tính và phương thức
 - không có các toán tử con trỏ
 - phép gán (=) không phải là phép toán copy nội dung đối tượng
- tham chiếu được lưu trữ trong vùng nhớ static/stack như các con trỏ trong C/C++



Toán tử New

- Phải tạo mọi đối tượng một cách tường minh bằng toán tử new
 - cấp phát vùng nhớ động
 - được tạo trong bộ nhớ Heap

- Ví dụ:

```
MyDate d;
```

```
MyDate birthday;
```

```
d = new MyDate();
```



Phép gán “=”

- Phép gán không phải là copy thông thường
 - copy nội dung của tham chiếu
 - 2 tham chiếu sẽ tham chiếu đến cùng đối tượng

```
Integer m = new Integer(10);  
Integer n  = new Integer(20);  
m = n;  
n.setValue(50);  
System.out.print(m);
```

“New” và “=”

```
MyDate d;
```

```
MyDate birthday;
```

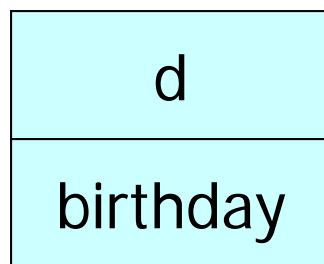
```
d = new MyDate(26, 9, 2005);
```

```
birthday = d;
```

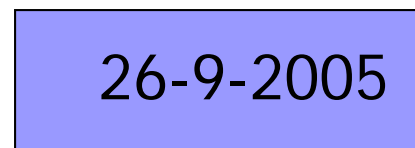
new operation

assign operation

Static/Stack memory



Heap memory



Toán tử quan hệ “==”

- So sánh nội dung của các dữ liệu kiểu nguyên thủy (int, long, float, ...)
- So sánh nội dung của tham chiếu chứ không so sánh nội dung của đối tượng do tham chiếu trỏ đến

```
Integer n1 = new Integer(47);
Integer n2 = new Integer(47);
System.out.println(n1 == n2);
System.out.println(n1 != n2);
--
false
true
```



So sánh nội dung đối tượng

```
class MyDate {  
    ...  
    boolean equalTo(MyDate d) {  
        ...  
    }  
}  
...  
MyDate d1 = new MyDate(10,10,1954);  
MyDate d2 = new MyDate(d1);  
System.out.println(d1.equalTo(d2));
```

Giải phóng bộ nhớ động (Garbage collection)

- Lập trình viên không cần phải giải phóng đối tượng
- JVM cài đặt cơ chế “Garbage collection” để giải phóng tự động các đối tượng không còn cần thiết
 - Tuy nhiên, *GC không nhất thiết hoạt động với mọi đối tượng* (không nhất thiết phải giải phóng bộ nhớ)
 - Không đảm bảo việc *phương thức hủy* luôn hoạt động
- GC tăng tốc độ phát triển và tăng tính ổn định của ứng dụng
 - Không phải viết mã giải phóng đối tượng
 - Do đó, *không bao giờ quên giải phóng đối tượng*

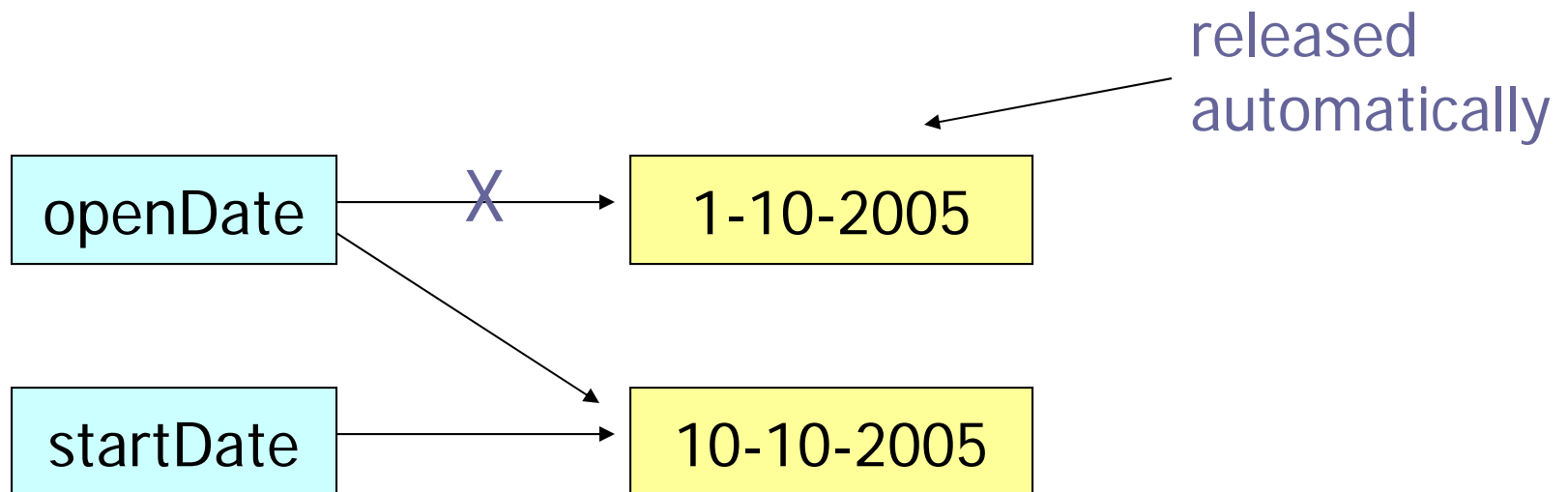


GC hoạt động như thế nào

- Sử dụng cơ chế đếm?
 - mỗi đối tượng có một số đếm các tham chiếu trỏ tới
 - giải phóng đối tượng khi số đếm = 0
- Giải phóng các đối tượng *chết*
 - kiểm tra tất cả các tham chiếu
 - đánh dấu các đối tượng còn được tham chiếu
 - giải phóng các đối tượng không được tham chiếu

Garbage Collection

```
MyDate openDate = new MyDate(1,10,2005);  
MyDate startDate = new MyDate(10,10,2005);  
openDate = startDate;
```





Truyền tham số và nhận giá trị trả lại

■ Truyền giá trị

- đối với dữ liệu kiểu nguyên thủy
- giá trị của tham số (RValue) được copy lên stack
- có thể truyền hằng số (vd: 10, 0.5, ...)

■ Truyền tham chiếu

- đối với đối tượng
- nội dung của tham chiếu (LValue) được copy lên stack



Truyền tham số trị

```
class MyDate {  
    ...  
    public boolean setYear(int y) {  
        ...  
    }  
    public int getYear() {  
        return year;  
    }  
    ...  
}  
...  
MyDate d = new MyDate();  
d.setYear(1975);  
int y = d.getYear();
```



Truyền tham chiếu

```
class MyDate {
    int year, month, day;
    public MyDate(int y, int m, int d) {
        year = y; month = m; day = d;
    }
    public void copy(MyDate d) {
        d.year = year;
        d.month = month;
        d.day = day;
    }
    public MyDate copy() {
        return new MyDate(day, month, year);
    }
    ...
}
```



Truyền tham chiếu

```
MyDate d1 = new MyDate(2005, 9, 26);  
MyDate d2 = new MyDate(2000, 1, 1);  
d1.copy(d2);  
MyDate d3 = new MyDate();  
d3 = d1.copy();
```



Tham chiếu `this`

- Java cung cấp tham chiếu `this` để trỏ tới chính đối tượng đang hoạt động
- `this` được sử dụng vào các mục đích như
 - tham chiếu tường minh đến thuộc tính và phương thức của đối tượng
 - truyền tham số và trả lại giá trị
 - dùng để gọi constructor

this làm giá trị trả lại

```
class Counter {
    private int c = 0;
    public Counter increase() {
        c++;
        return this;
    }
    public int getValue() {
        return c;
    }
}
...
Counter count = new Counter();
System.out.println(count.increase().increase().getValue())
;
```



this làm tham số


```
class Document {
    Viewer vi;
    ...
    Document(Viewer v) {
        vi = v;
        ...
    }
    void display() {
        vi.display(this);
    }
    ...
}
```


Gọi constructor bằng `this`

```
class MyDate {
    private int year, month, day;

    public MyDate(int y, int m, int d) {
        ...
    }
    // copy constructor
    MyDate(MyDate d) {
        this(d.year, d.month, d.day);
        System.out.println("copy constructor called");
    }
    ...
}
```

- Constructor chỉ được gọi bên trong một constructor khác và chỉ được gọi một lần ở thời điểm (vị trí) đầu tiên.



Phương thức và thuộc tính static

- Có thể khai báo phương thức và thuộc tính là tĩnh (static)
 - độc lập với đối tượng
 - có thể sử dụng mà không cần có đối tượng
- Phương thức tĩnh
 - không sử dụng được thuộc tính thông thường (non-static)
 - không gọi được các phương thức thông thường

Ví dụ: Hello.java

```
public class Hello{
    static Hello h = new Hello();
    sayHello() {System.out.println("Hello, world");}
    static Hello getRef() { return h;}
    public static void main (String[] args) {
        h.sayHello();
        Hello h2 = new Hello();
        h2.sayHello();
        // sayHello();
        getRef().sayHello();
    }
}
```


Gói các lớp đối tượng (package)

- Các lớp đối tượng được chia thành các gói
 - nếu không khai báo thì các lớp thuộc gói default
 - các lớp trong cùng một tệp mã nguồn luôn thuộc cùng một gói
- Tồn tại mức truy cập package
 - mức package là mặc định (nếu không khai báo tường minh là public hay private)
 - các đối tượng của các lớp thuộc cùng gói có thể truy cập đến non-private members của nhau
 - chỉ có thể tạo (new) đối tượng của lớp được khai báo là public của gói khác



Hello.java:

```
class HelloMsg {  
    void sayHello() {  
        System.out.println("Hello, world!");  
    }  
}  
  
public class Hello {  
    public static void main(String[] args) {  
        HelloMsg msg = new HelloMsg();  
        msg.sayHello();  
    }  
}
```



Khai báo và sử dụng package

- Khai báo gói bằng lệnh `package`
 - các gói được lưu trữ theo cấu trúc cây thư mục
 - sử dụng tham số `-d` để tạo thư mục khi biên dịch
- Dùng lệnh `import` để khai báo việc sử dụng một gói đã có



HelloMsg.java:

```
package hanv;  
  
public class HelloMsg {  
    public void sayHello() {  
        System.out.println("Hello, world!");  
    }  
}
```



Hello.java:

```
import hanv>HelloMsg;

public class Hello {
    public static void main(String[] args) {
        HelloMsg msg = new HelloMsg();
        msg.sayHello();
    }
}
```




Biên dịch & thực hiện

■ Biên dịch

```
javac HelloMsg.java -d .
```

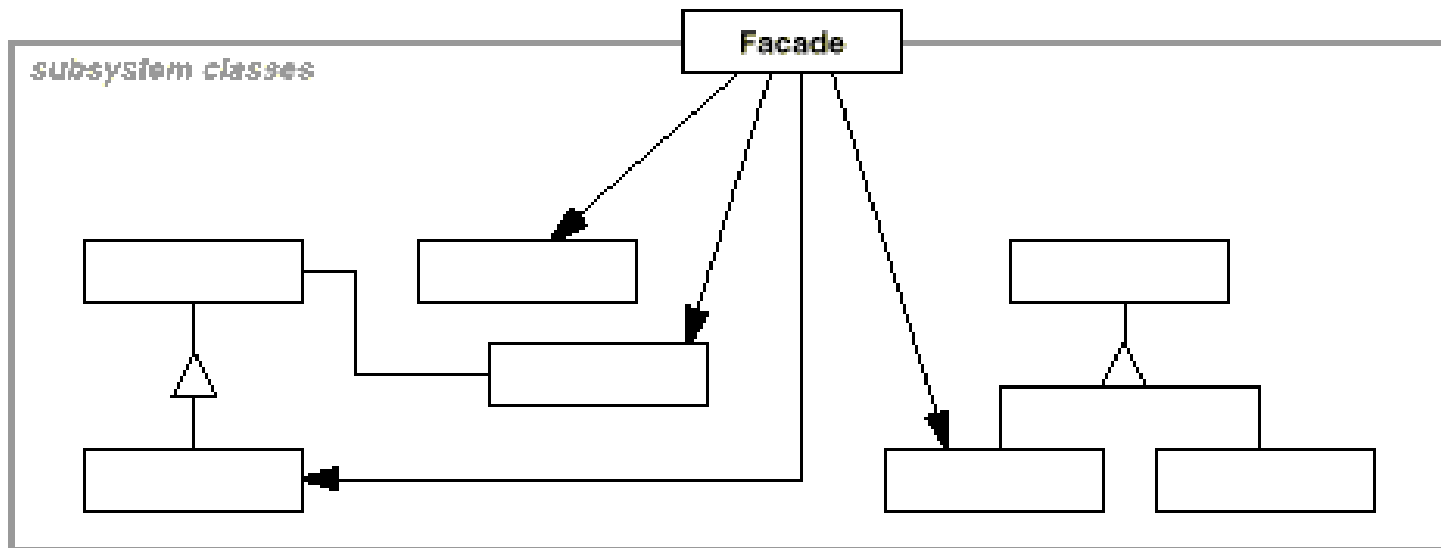
```
javac Hello.java
```

■ Thực hiện

```
java Hello
```

Package & Façade pattern

- Che giấu cấu trúc nội tại của gói
- Truy cập thông qua *đối tượng thuộc lớp giao diện*





Đối tượng hợp thành (Composition)

- Đối tượng có thể chứa các đối tượng khác (các thuộc tính không thuộc kiểu nguyên thủy)
- Thuộc tính là tham chiếu phải được tạo ra bằng new hoặc được gán cho một đối tượng đã tồn tại

```
class Person {  
    private String name;  
    private MyDate birthday = new MyDate(1,1,2000);  
    ...  
}
```

Get/Set thuộc tính không thuộc kiểu nguyên thủy

```
class Person {
```

```
...
```

```
    public MyDate getBirthday() {  
        return birthday;  
    }
```

```
}
```

```
Person p = new Person(...);
```

```
MyDate d = p.getBirthday();
```

```
d.setYear(1900);
```



Get/Set bằng copy constructor

```
class Person {
    private String name;
    private MyDate birthday;
    public Person(String s, MyDate d) {
        name = s;
        birthday = new MyDate(d);
    }
    public MyDate getBirthday() {
        return new MyDate(birthday);
    }
    public void setBirthday(MyDate d) {
        birthday = new MyDate(d);
    }
    ...
}
```



Vào ra từ luồng dữ liệu chuẩn

- Luồng ra chuẩn: `System.out`
 - xuất ra luồng ra chuẩn (standard output)
 - có thể tái định hướng
- Luồng thông báo lỗi: `System.err`
 - xuất ra **Console** (thiết bị output chuẩn)
 - không thể tái định hướng
- Luồng dữ liệu vào chuẩn: `System.in`
 - chưa sẵn sàng cho sử dụng



Nhập dữ liệu từ luồng vào chuẩn

- **InputStream**: lớp đối tượng ứng với luồng vào chuẩn
 - **System.in**: đối tượng tương ứng
 - chưa có phương thức nhập dữ liệu
- **Scanner**: nhập dữ liệu kiểu nguyên thủy và chuỗi ký tự
 - **next**: nhập chuỗi ký tự
 - **nextType** : nhập một dữ liệu kiểu Type
 - **hasNext**, **hasNextType** : kiểm tra xem còn dữ liệu không.



Ví dụ

```
Scanner sc = new Scanner(System.in);
```

```
System.out.println(sc.next());
```

```
int i = sc.nextInt();
```

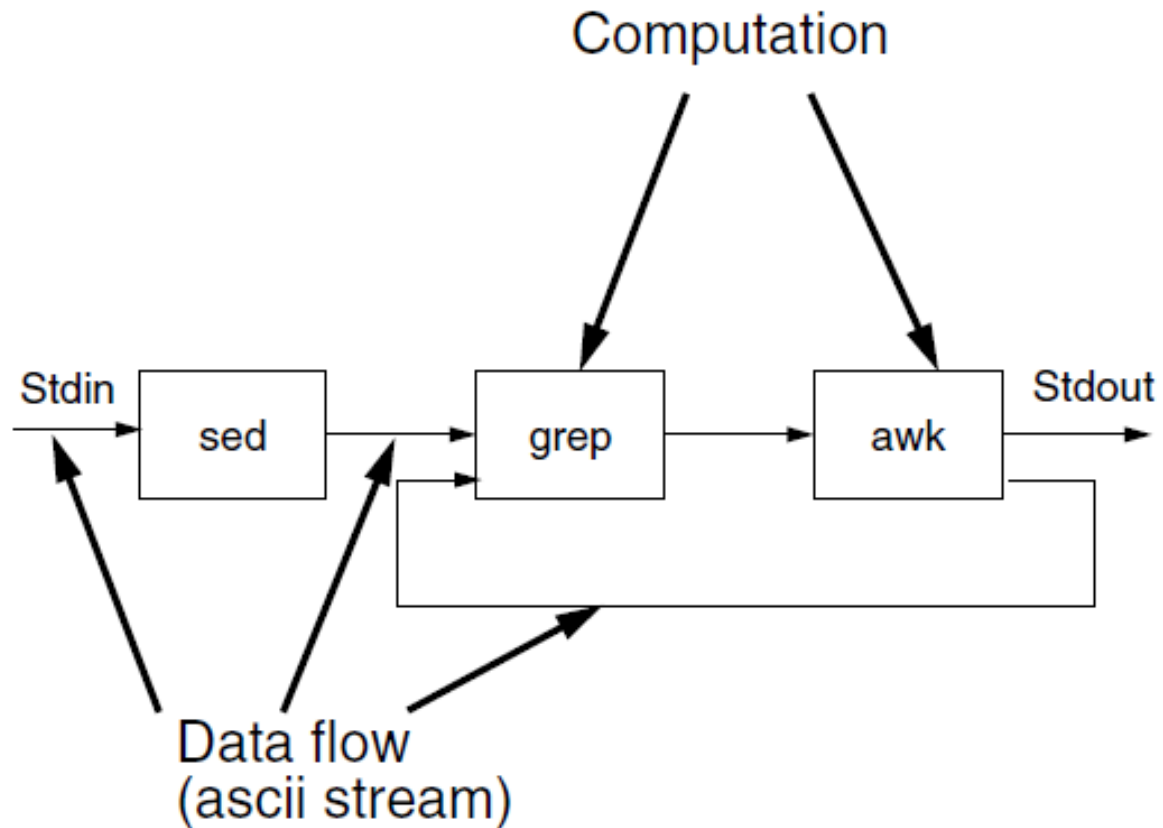
```
while (sc.hasNextLong()) {  
    long aLong = sc.nextLong();  
}
```



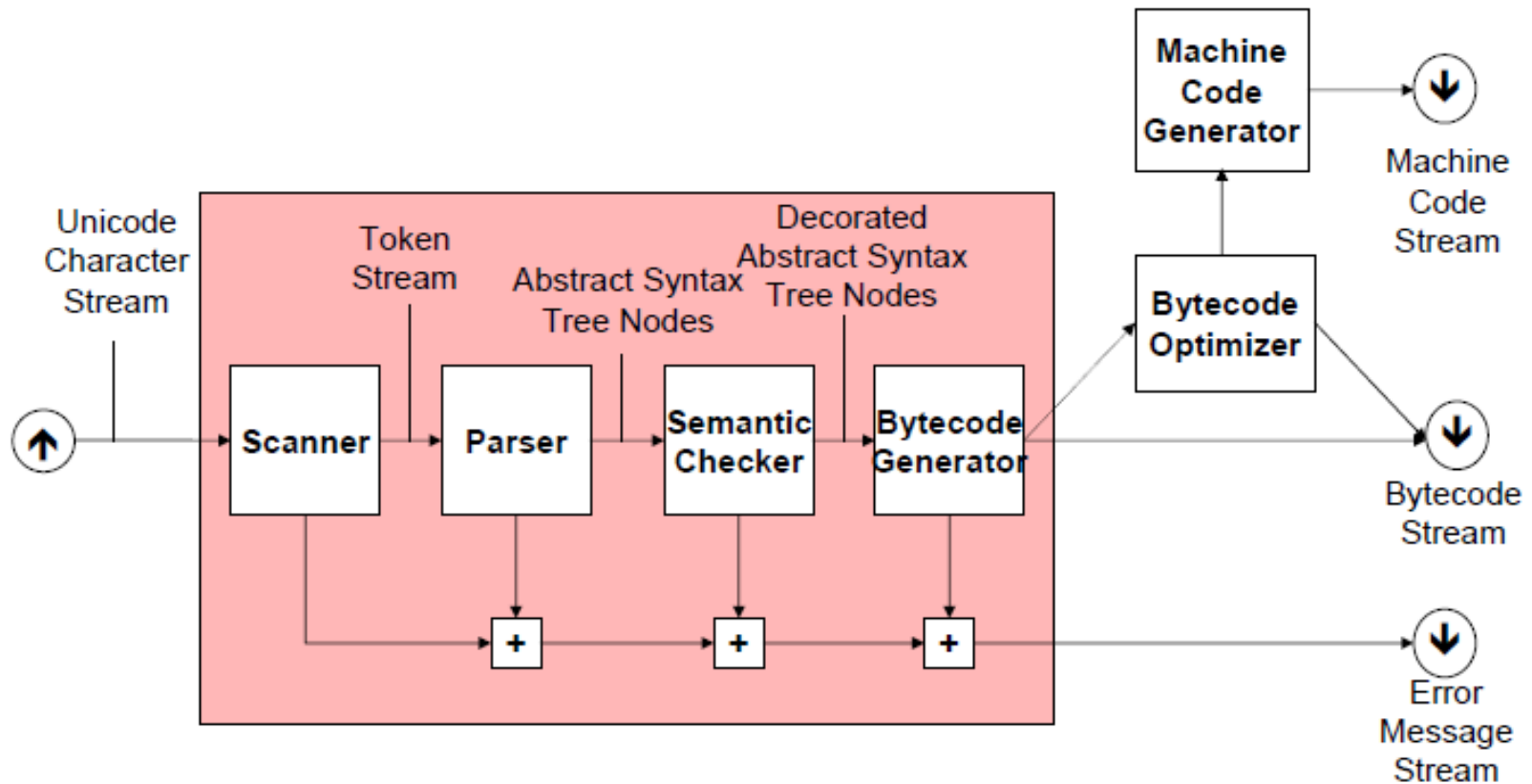

Định hướng lại luồng vào/ra và mẫu kiến trúc Pipe & Filter

- Các luồng vào ra có thể định hướng từ các nguồn/đích dữ liệu khác nhau
 - Tập, chương trình, thiết bị (máy in,...)
- Có thể xây dựng các phần mềm phức tạp từ việc ghép nối các phần mềm nhỏ hơn
 - Tái sử dụng lại cao
 - Ghép nối động

Pipe and Filter



Pipe and Filter





Tham số dòng lệnh

CmdLineParas.java:

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

Ví dụ:

```
#java CmdLineParas hello world  
hello  
world
```



Bài tập: Xây dựng lớp Singleton

- Không thể sinh ra nhiều hơn một thực thể của lớp trong một lần chạy chương trình
 - Có thể có nhiều tham chiếu, nhưng chỉ có một đối tượng (một vùng bộ nhớ được cấp phát)
- Thường ứng dụng trong quản lý tài nguyên hệ thống
 - Để tránh xung đột về tài nguyên của các tiến trình tương tranh (song song)



Bài tập: Cấu trúc dữ liệu

- Xây dựng Danh sách liên kết đơn
 - Thêm phần tử vào đầu/cuối danh sách
 - Bớt phần tử khỏi danh sách
 - Tìm phần tử theo khóa
- Xây dựng Cây tìm kiếm nhị phân
 - Tìm phần tử theo khóa
 - Thêm, bớt phần tử
 - Cân bằng cây