2110211 โครงสร้างข้อมูลเบื้องต้น

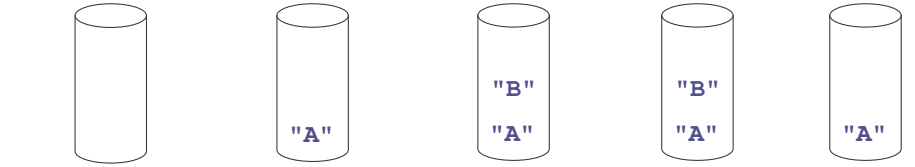
Stack

สมชาย ประสิทธิ์จูตระกูล

กองซ้อน : stack

```
public interface Stack {
  public boolean isEmpty();
  public int size();
  public void push(Object e);
  public Object peek();
  public Object pop();
}
```

LIFO
Last-In First-Out



Stack s = new ArrayStack()　s.push("A")　s.push("B")　s.peek() "B"　s.pop() "B"

Stack คล้าย List

- Stack คือ list ที่เราเพิ่มและลบข้อมูลที่ปลายด้านเดียว
- สร้าง Stack ด้วย list แบบง่าย ๆ

```
public class ArrayListStack implements Stack {
  private ArrayList list = new ArrayList();

  public boolean isEmpty() {return list.isEmpty();}

  public int size() {return list.size();}

  public void push(Object e) {list.add(list.size(),e);}

  public Object peek() {
    if (isEmpty()) throw new NoSuchElementException();
    return list.get(list.size()-1);}

  public Object pop() {
    Object e = peek();
    list.remove(list.size()-1);
    return e;
  }
}
```

Composition + Delegation

หนึ่ง stack สองออปเจกต์ เสียเวลาเรียกบริการให้ list ทำต่อ

ArrayStack : สร้าง Stack ด้วยอาเรย์

- คล้าย ArrayCollection
- มีเพิ่มกับลบเฉพาะที่ด้านท้าย
- ขยายขนาดของอาเรย์ได้ เมื่อเต็ม

ArrayStack s;	size	elementData
s = new ArrayStack();	0	
s.push("A");	1	A
s.push("B");	2	A B
s.push("C");	3	A B C
s.pop();	2	A B

```
public class ArrayStack implements Stack {
  private Object[] elementData;
  private int size;

  public ArrayStack() { elementData = new Object[1]; }
  public boolean isEmpty() { return size == 0; }
  public int size() { return size; }
  public void push(Object e) {
    if (size == elementData.length) {
      Object[] newA = new Object[2*elementData.length];
      for(int i=0; i<elementData.length; i++)
        newA[i] = elementData[i];
      elementData = newA;
    }
    elementData[size++] = e;
  }
  public Object peek() {
    if (isEmpty()) throw new NoSuchElementException ();
    return elementData[size-1];
  }
  public Object pop() {
    Object e = peek();
    elementData[--size] = null;
    return e;
  }
}
```

ตัวอย่างการใช้งาน Stack

- การตรวจสอบการใส่วงเล็บ ( ) { } [ ] ...
- postfix :
  - การหาค่าของนิพจน์ postfix
  - การแปลงนิพจน์ infix เป็น postfix
- jvm
  - การเรียกเมท็อด
  - การจัดการที่เก็บข้อมูลชั่วคราวของ jvm
- ...

การตรวจสอบการใส่วงเล็บ

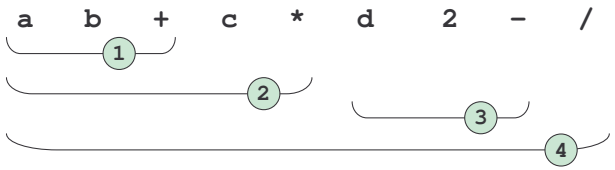
- ถูก : **({ ( ) [{ }] })**
- ผิด : เปิดปิดไม่ตรงกัน (**{**) **]**
- ผิด : มีเปิดมากไป **(** { ( ) }
- ผิด : มีปิดมากไป ({ ( ) }) **) }**
- วิธีทำ
  - อ่านมาทีละตัว
  - ถ้าเป็นวงเล็บเปิด ให้ push ลง stack
  - ถ้าเป็นวงเล็บปิด ให้ pop จาก stack มาตรวจสอบว่าเป็นวงเล็บเปิดที่ตรงกันวงเล็บปิดที่พบหรือไม่
  - เมื่อใดอยาก pop ถ้า isEmpty แสดงว่า ปิดมีมากไป
  - เมื่ออ่านเสร็จหมด stack ยังมีข้อมูล แสดงว่า เปิดมีมากไป


```
public static void main(String[] args) throws IOException {
   BufferedReader br = new BufferedReader( new FileReader("g.txt"));
   String line;
   String open = "{([", String close = "})]", openclose = open + close;
   Stack s = new ArrayStack();
   while ((line = br.readLine()) != null) {
     StringTokenizer st = new StringTokenizer(line, openclose, true);
     while (st.hasMoreTokens()) {
       String token = st.nextToken();
       if (open.indexOf(token) >= 0) { // วงเล็บเปิด ?
         s.push(token);
       } else {
         int k = close.indexOf(token); // วงเล็บปิด ?
         if (k >= 0) {
           if (s.isEmpty()) error( "มีวงเล็บปิดมากไป" );
           if (!s.pop().equals(open.substring(k, k + 1))) // คู่กัน ?
             error( "วงเล็บเปิดปิดไม่ตรงกัน" );
         }
       }
     }
   }
   br.close();
   if (!s.isEmpty()) error( "มีวงเล็บเปิดมากไป" );
   System.out.println("ok");
 }

 private static void error(String s) {
   System.out.println("error : " + s);
   System.exit(-1);
 }
```

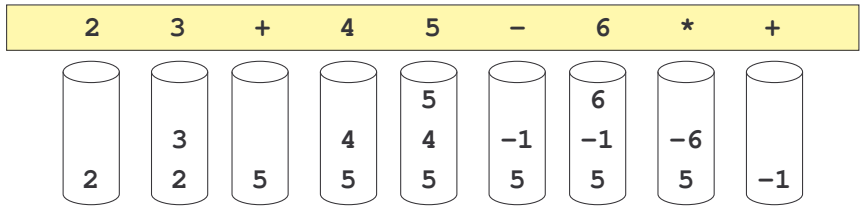
นิพจน์ Infix และ Postfix

- infix
  - a + b * c / d – 2, (a + b) * c / (d – 2)
  - ต้องกำหนดลำดับการทำก่อนหลังของ operators
  - ใช้วงเล็บช่วย
- postfix
  - a b c * d / + 2 – , a b + c * d 2 - /
  - ลำดับการทำงานของ operators คือจาก ซ้ายไปขวา
  - ไม่จำเป็นต้องมีวงเล็บ



Evaluating Postfix Expression

- 2 3 + 4 5 – 6 * + มีค่าเท่าไร ?
- สามารถใช้ stack ช่วยหาค่าของนิพจน์ postfix
- วิธีทำ
  - ดูทีละตัวใน postfix จากซ้ายไปขวา
  - ถ้าเป็น operand ให้ push ของ stack
  - ถ้าเป็น operator ให้ pop operands จาก stack ตามที่ operator ต้องการมาประมวลผล แล้ว push ผลลัพธ์
  - ทำเสร็จ คำตอบจะอยู่ที่ top of stack



การหาค่าของนิพจน์ postfix

```
public class Postfix {
  static enum OP {ADD, SUB, MUL, DIV};

  public static void main(String[] args) {
    List postfix = new ArrayList();
    postfix.add(new Double(2));
    postfix.add(new Double(3));
    postfix.add(OP.ADD);
    postfix.add(new Double(4));
    postfix.add(new Double(5));
    postfix.add(OP.SUB);
    postfix.add(new Double(6));
    postfix.add(OP.MUL);
    postfix.add(OP.DIV);
    System.out.println(evalPostfix(postfix));
  }
  ...
```

2 3 + 4 5 – 6 * +


```
static double evalPostfix(List postfix) {
  Stack s = new ArrayStack();
  for (int i = 0; i < postfix.size(); i++) {
    Object token = postfix.get(i);
    if (!(token instanceof OP)) {
      s.push(token);
    } else {
      if (s.isEmpty()) break;
      double v2 = ((Double)s.pop()).doubleValue();
      if (s.isEmpty()) break;
      double v1 = ((Double)s.pop()).doubleValue();
      double v = 0;
      if (token == OP.ADD) v = v1 + v2;
      else if (token == OP.SUB) v = v1 – v2;
      else if (token == OP.MUL) v = v1 * v2;
      else if (token == OP.DIV) v = v1 / v2;
      else assert false;
      s.push(new Double(v));
    }
  }
  if (s.size() != 1) throw new IllegalArgumentException();
  return ((Double)s.pop()).doubleValue();
}
```

```
static enum OP {
  ADD {double eval(double d1, double d2) {return d1 + d2;} },
  SUB {double eval(double d1, double d2) {return d1 - d2;} },
  MUL {double eval(double d1, double d2) {return d1 * d2;} },
  DIV {double eval(double d1, double d2) {return d1 / d2;} };
  abstract double eval(double d1, double d2);
}

static double evalPostfix(List postfix) {
  Stack s = new ArrayStack();
  for (int i = 0; i < postfix.size(); i++) {
    Object token = postfix.get(i);
    if (!(token instanceof OP)) {
      s.push(token);
    } else {
      if (s.isEmpty()) break;
      double v2 = ((Double)s.pop()).doubleValue();
      if (s.isEmpty()) break;
      double v1 = ((Double)s.pop()).doubleValue();
      s.push(new Double( ((OP) token).eval(v1,v2) ));
    }
  }
  if (s.size() != 1) throw new IllegalArgumentException();
  return ((Double)s.pop()).doubleValue();
}
```

ใช้ enum ของ Java 5 ให้เต็มที่ โดยย้าย operations ไปอยู่ใน enum OP

การแปลงนิพจน์ infix เป็น postfix

- List infix : < 2, +, 3, * 4 >
- List postfix : < 2, 3, 4, * + >

```
public static List infixToPostfix( List infix ) {
  List postfix = new ArrayList();
  Stack s = new ArrayStack();
  for (int i = 0; i < infix.size(); ++i) {
    Object token = infix.get(i);
    ...
    postfix.add(???);
    ...
  }
  ...
  return postfix;
}
```

ใช้ stack ช่วยในการแปลง

การแปลงนิพจน์ infix เป็น postfix

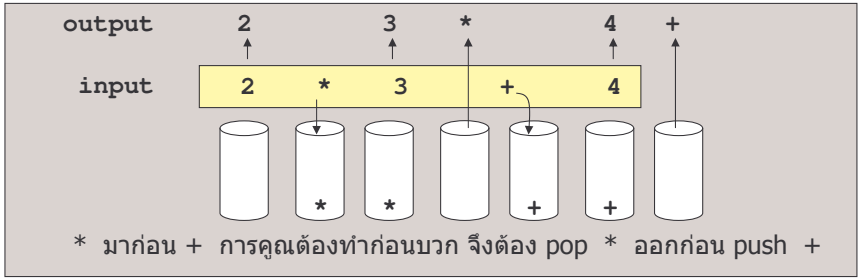
```
public static List infixToPostfix(List infix) {
  List postfix = new ArrayList();
  Stack s = new ArrayStack();
  for (int i = 0; i < infix.size(); ++i) {
    Object token = infix.get(i);
    if (!(token instanceof OP)) {
      postfix.add(token);
    } else {
      while( ??? ) {
        postfix.add(s.pop());
      }
      s.push(token);
    }
  }
  while (!s.isEmpty()) postfix.add(s.pop());
  return postfix;
}
```

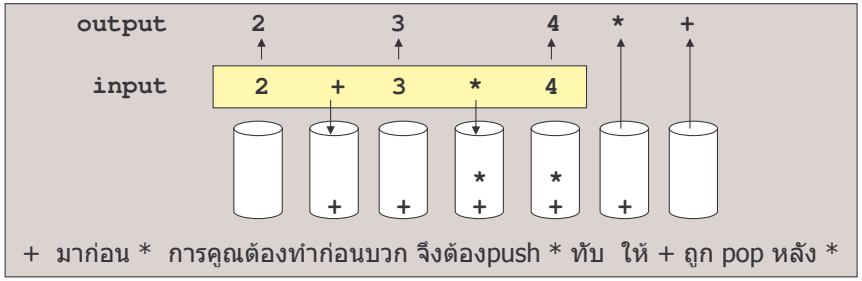
ดูที่ละตัว

ถ้าเป็น operand, เพิ่มต่อในผลลัพธ์

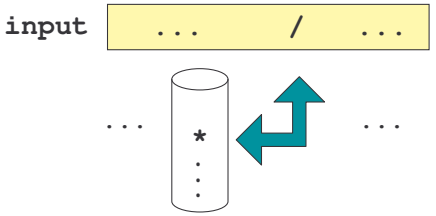
ถ้าเป็น operator
อาจ pop operators ใน stack ออกเป็นผลลัพธ์
ตามด้วยการ push operator ตัวใหม่

การแปลงนิพจน์ infix เป็น postfix

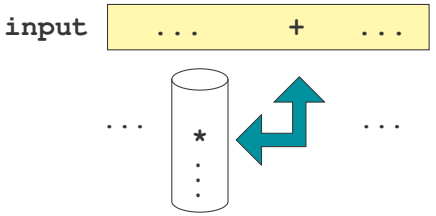




operator ที่ top of stack กับ operator ตัวใหม่



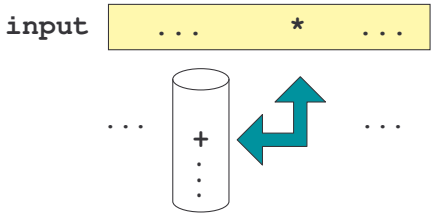
- * มาก่อน / (จึงอยู่ใน stack)
- * มี priority เท่ากับ /
- จึงต้องให้ทำ * ก่อน /
- pop * ออกไปที่ output



- * มาก่อน - (จึงอยู่ใน stack)
- * มี priority มากกว่า –
- จึงต้องให้ทำ * ก่อน +
- pop * ออกไปที่ output

pop เมื่อ priority(operator ที่ top of stack) ≥ priority(operator ตัวใหม่)

operator ที่ top of stack กับ operator ตัวใหม่



- + มาก่อน * (จึงอยู่ใน stack)
- + มี precedence น้อยกว่า *
- + ต้องรอให้ * ทำก่อน
- ยังไม่ pop + ออกไปที่ output

ไม่ pop เมื่อ priority(operator ที่ top of stack) < priority(operator ตัวใหม่)

```
} else {
  OP op = (OP) token;
  while (!s.isEmpty() &&
         ((OP) s.peek()).priority >= op.priority) {
    postfix.add(s.pop());
  }
  s.push(token);
}
```


เพิ่ม priorities ของ operator ให้กับ enum OP

```
static enum OP {
  ADD(10), SUB(10), MUL(30), DIV(30);

  int priority;
  OP(int p) { priority = p; }
}
```

คลาส OP มี 4 ออปเจกต์ แต่ละออปเจกต์เก็บ priority ต่างกัน

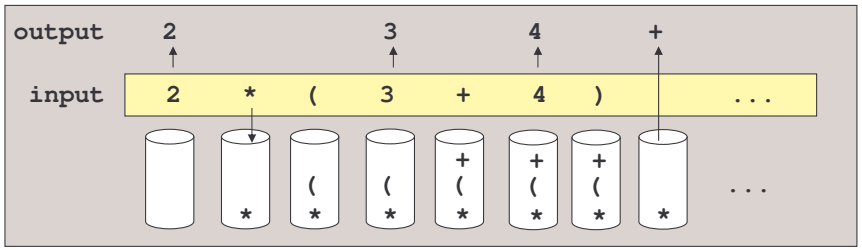
```
static enum OP {
  ADD(10) {double eval(double d1, double d2) {return d1 + d2;} },
  SUB(10) {double eval(double d1, double d2) {return d1 - d2;} },
  MUL(30) {double eval(double d1, double d2) {return d1 * d2;} },
  DIV(30) {double eval(double d1, double d2) {return d1 / d2;} };

  abstract double eval(double d1, double d2);
  int priority;
  OP(int p) { priority = p; }
}
```

OP เป็น abstract class มี 4 subclasses สร้างออปเจกต์ให้ subclass ละตัว แต่ละ subclass ต่างกันตรงที่เมท็อด eval

ซับซ้อนขึ้นเมื่อมีวงเล็บใน infix

- ภายในวงเล็บเสมือนเป็นนิพจน์ย่อย ที่ต้องทำก่อน
  - พบวงเล็บเปิด คือเริ่มการแปลงนิพจน์ย่อยใหม่
  - พบวงเล็บปิด คือจบการแปลงนิพจน์ย่อย
- พิจารณาวงเล็บเปิดให้เสมือนเป็น operator



- พบวงเล็บเปิดต้องถูก push เสมอ
- พออยู่ใน stack วงเล็บเปิดมี priority ต่ำมาก ถูกทับด้วย operators ทุกแบบที่ตามมา
- พบวงเล็บปิด จบนิพจน์ย่อย ต้องลุย pop ออก output จนพบ วงเล็บเปิด

การแปลง infix (มีวงเล็บได้) เป็น postfix

```
if (!(token instanceof OP)) {
  postfix.add(token);
} else {
  OP op = (OP) token;
  if ( op == OP.CLOSEPAREN ) {
    while(s.peek() != OP.OPENPAREN) {
      postfix.add(s.pop());
    }
    s.pop(); // pop the open paren
  } else {
    while( op != OP.OPENPAREN &&  !s.isEmpty() &&
          ((OP) s.peek()).priority >= op.priority ) {
      postfix.add(s.pop());
    }
    s.push(token);
  }
}
```

```
static enum OP {
  ADD(10), SUB(10), MUL(30), DIV(30),
  OPENPAREN(0), CLOSEPAREN(0);
  ...
}
```


การบ้าน : ยกกำลัง

- ใน Java เครื่องหมาย ^ แทน xor
- ขอใช้ ^ แทนการยกกำลัง (แบบภาษา Basic)
  - 3^2 มีค่าเท่ากับ 9
  - โดยทั่วไปถ้าเขียน 3^2^3 มีค่าเท่ากับ 3^(2^3)

$$3^{2^3} = 3^{\left(2^3\right)}$$

  - แสดงว่า ^ จะทำแบบขวามาซ้าย
- ดังนั้น
  - infix : (6 ^ 4) ^ 2 + 5 ^ 3 ^ 2 คือ $\left(6^4\right)^2 + 5^{3^2}$
  - postfix : 6 4 ^ 2 ^ 5 3 2 ^ ^ +
- จงปรับ infixToPostfix และ OP ให้รู้จัก ^

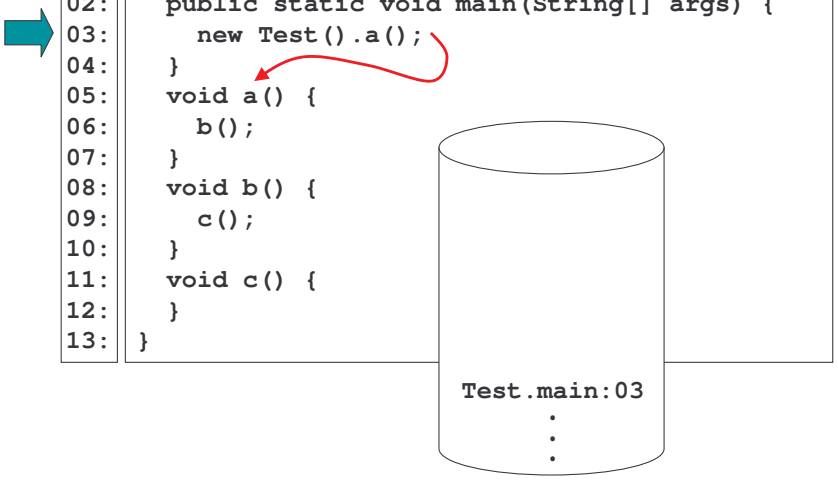
ขั้นตอนการเรียกเมท็อด

```
01: public class Test{
02:   public static void main(String[] args) {
03:     new Test().a();
04:   }
05:   void a() { b(); }
06:   void b() { c(); }
07:   void c() {
08:     try {
09:       int a = 0;
10:       int b = 3 / a;
11:     } catch( Exception e ) {
12:       e.printStackTrace();
13:     }
14:   }
15: }
```

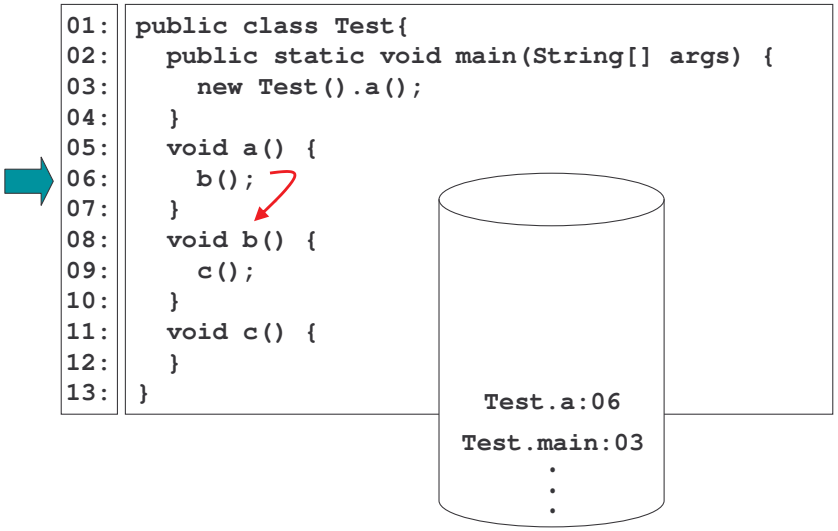
```
java.lang.ArithmeticException: / by zero
        at Test.c(Test.java:10)
        at Test.b(Test.java:6)
        at Test.a(Test.java:5)
        at Test.main(Test.java:3)
```


jvm ใช้ stack เก็บสถานะการเรียกเมท็อด

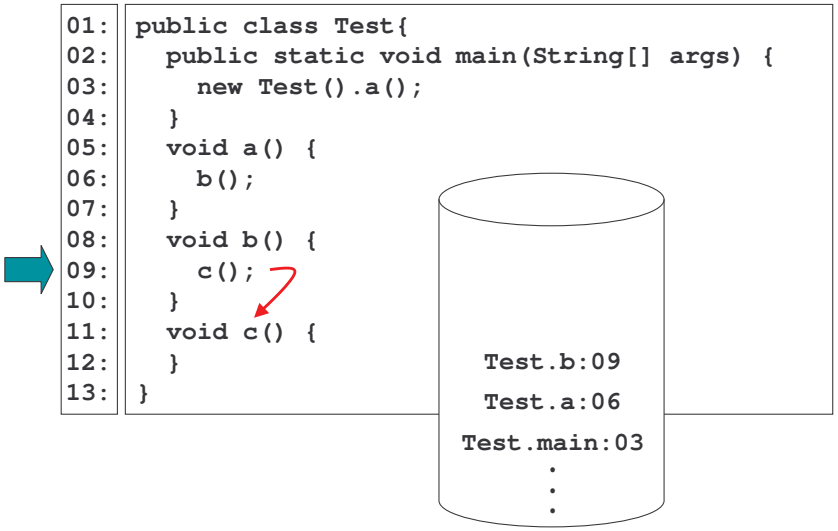
```
01: public class Test{
02:   public static void main(String[] args) {
03:     new Test().a();
04:   }
05:   void a() {
06:     b();
07:   }
08:   void b() {
09:     c();
10:   }
11:   void c() {
12:   }
13: }
```



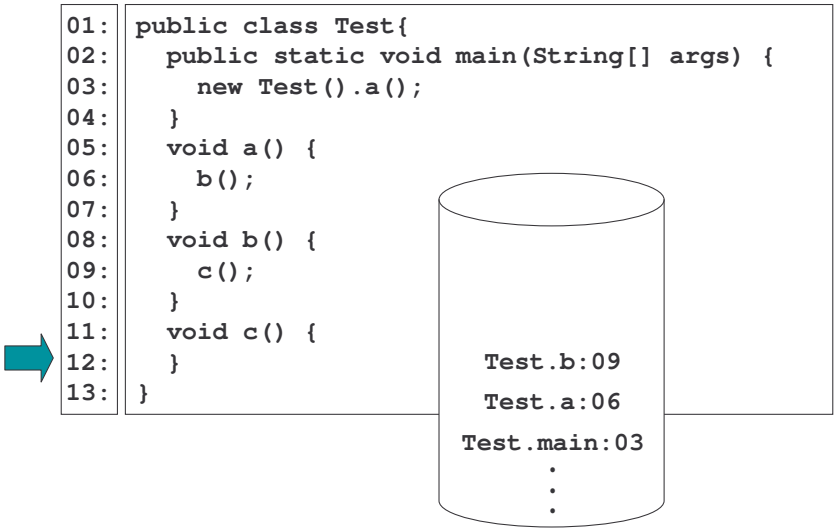
jvm ใช้ stack เก็บสถานะการเรียกเมท็อด



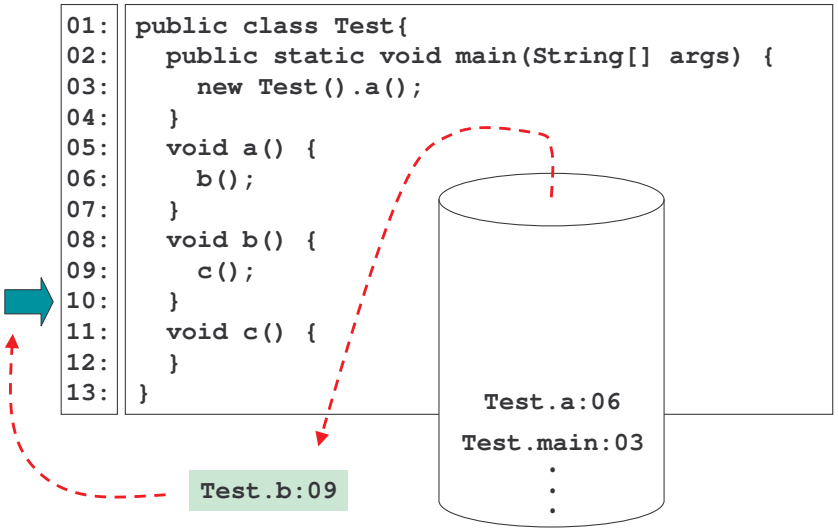
jvm ใช้ stack เก็บสถานะการเรียกเมท็อด



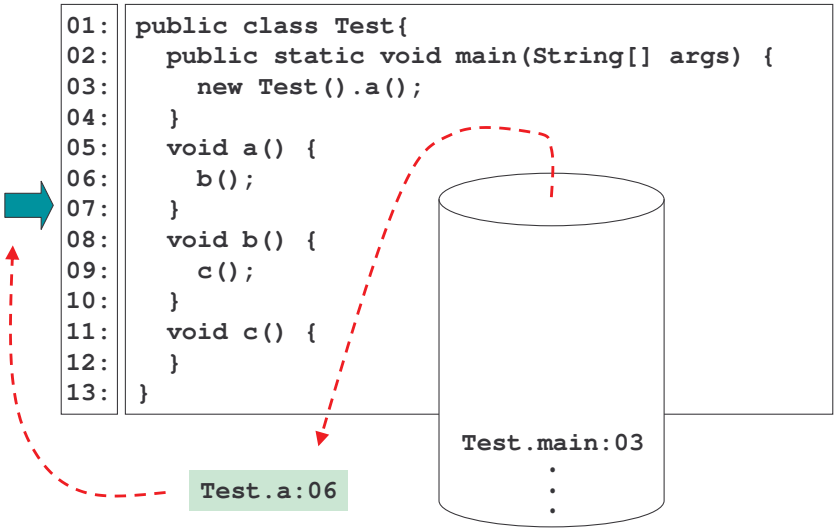
jvm ใช้ stack เก็บสถานะการเรียกเมท็อด



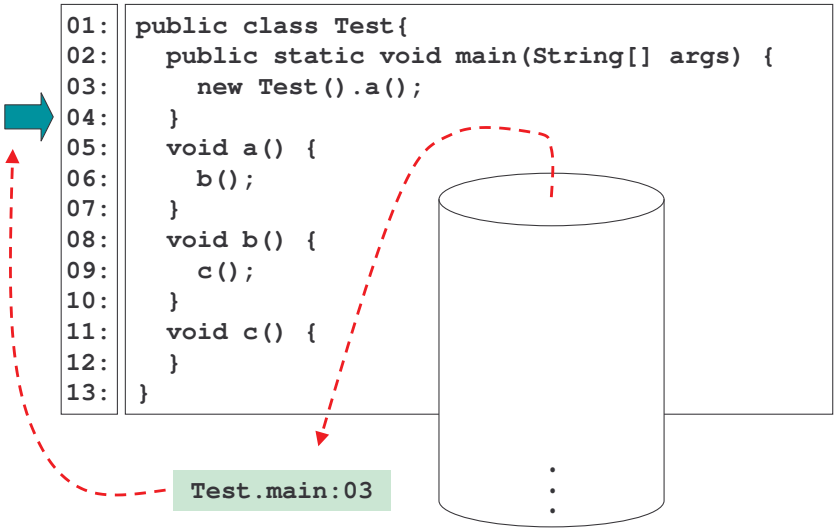
jvm ใช้ stack เก็บสถานะการเรียกเมท็อด



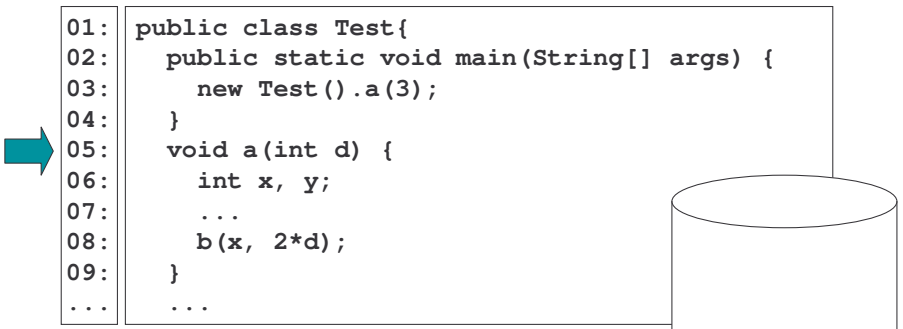
jvm ใช้ stack เก็บสถานะการเรียกเมท็อด



jvm ใช้ stack เก็บสถานะการเรียกเมท็อด



jvm ใช้ stack เก็บข้อมูลชั่วคราวของเมท็อด

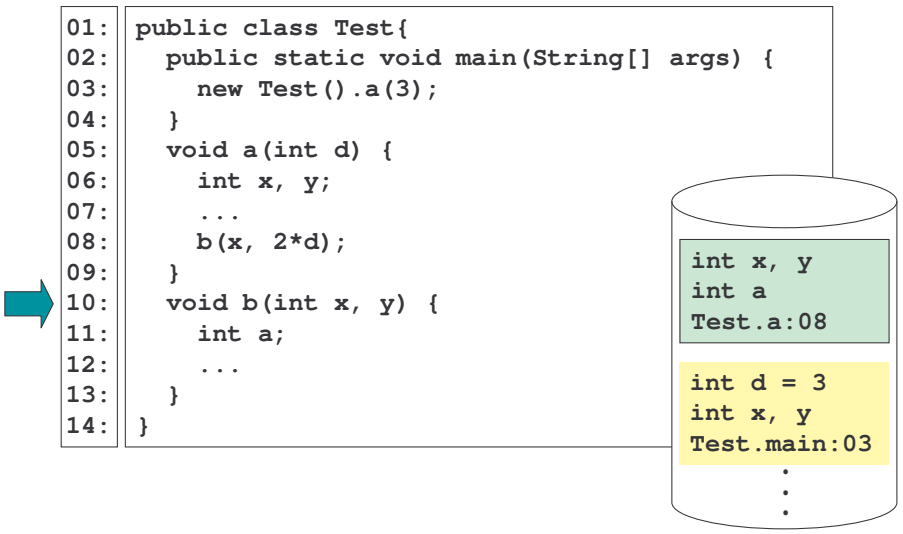


parameters และ local variables ต่าง ๆ ภายในเมท็อด ใช้ได้เฉพาะภายในเมท็อด

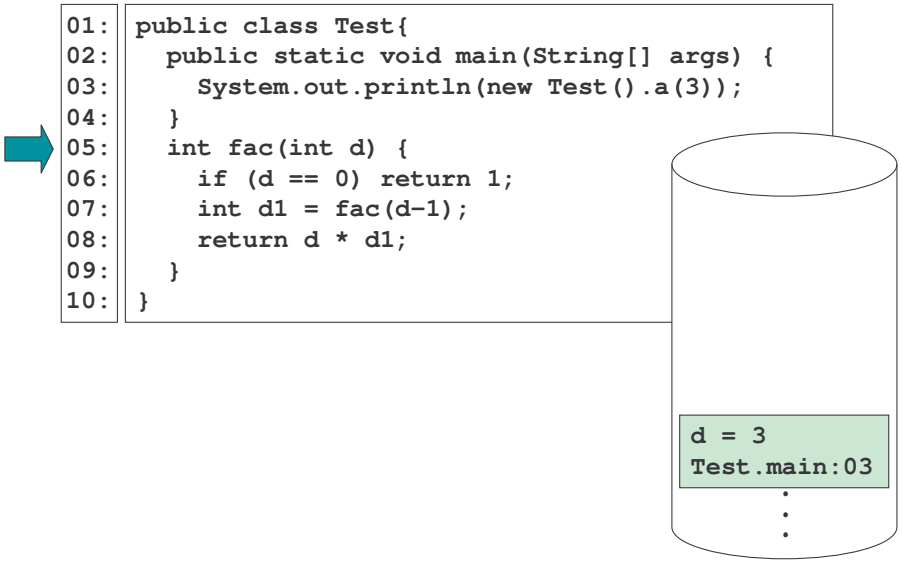
เสมือนว่าตัวแปร d, x, y เกิด เมื่อเมท็อด a ถูกเรียก และ หายไปเมื่อ return ออกจาก a

ตัวแปรเหล่านี้ถูกเก็บใน stack

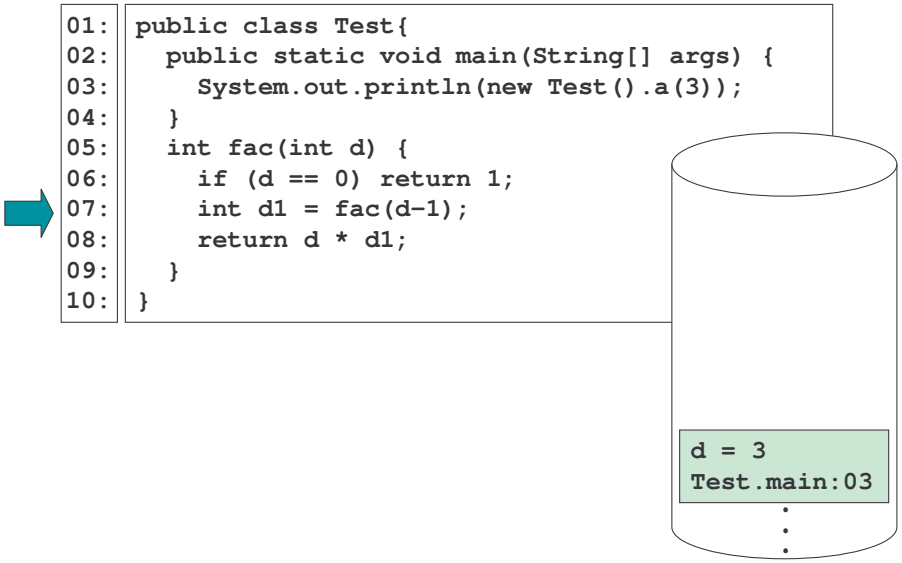
jvm ใช้ stack เก็บข้อมูลชั่วคราวของเมท็อด



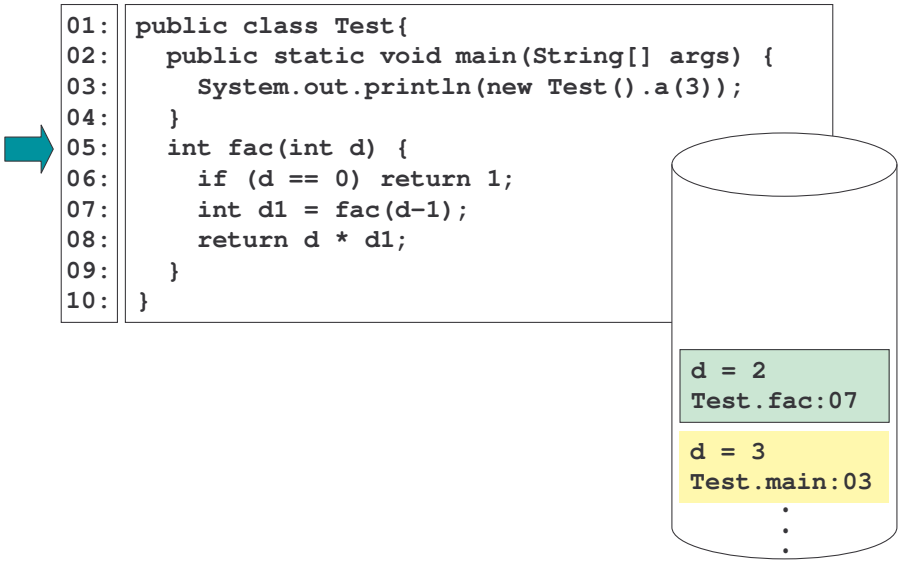
jvm ใช้ stack จัดการ recursive calls



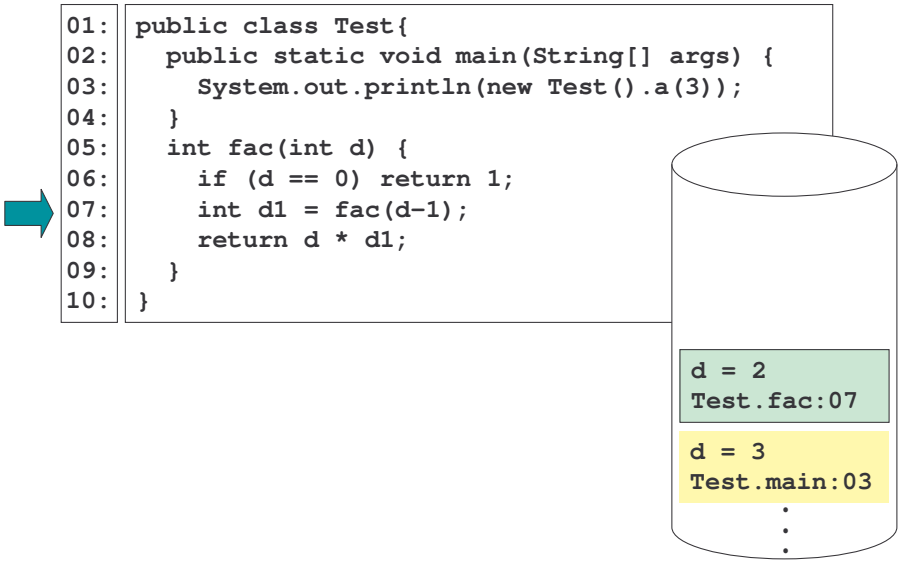
jvm ใช้ stack จัดการ recursive calls



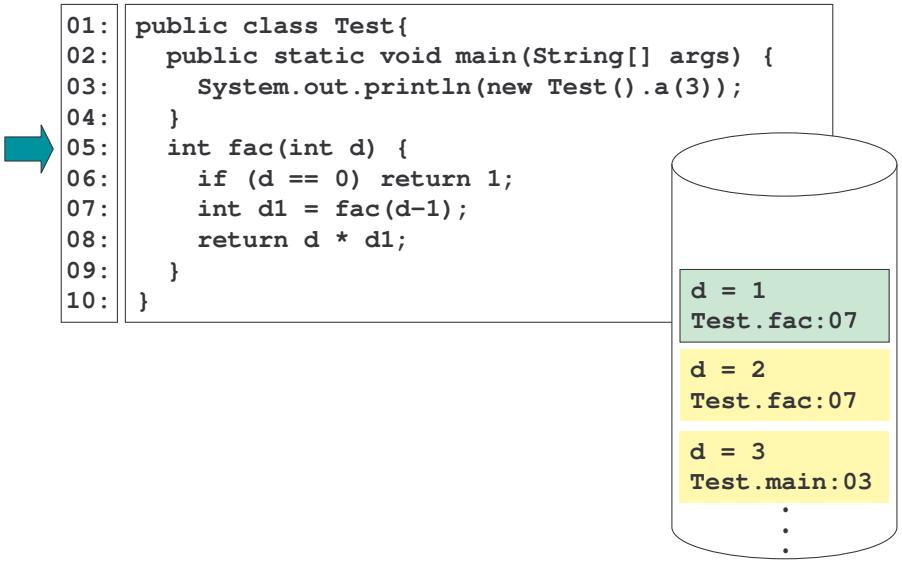
jvm ใช้ stack จัดการ recursive calls



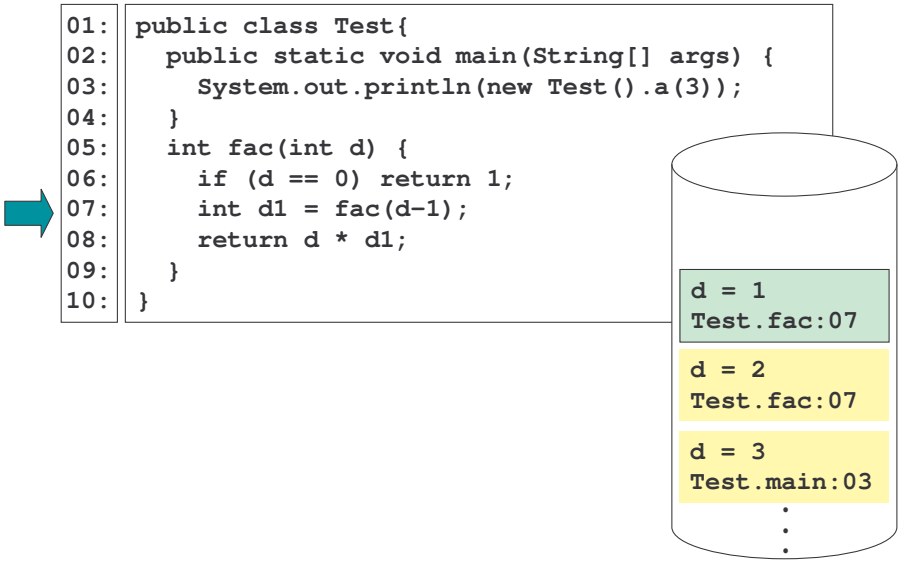
jvm ใช้ stack จัดการ recursive calls



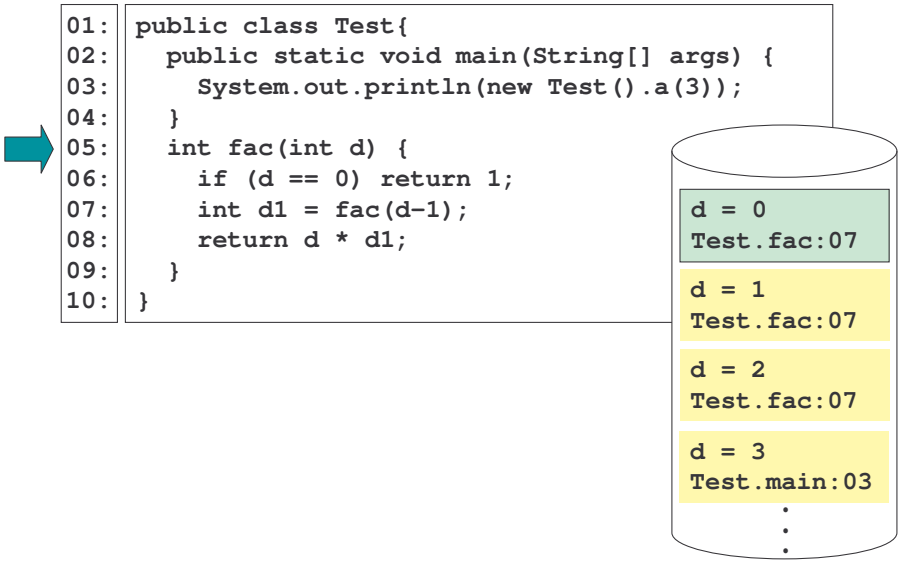
jvm ใช้ stack จัดการ recursive calls



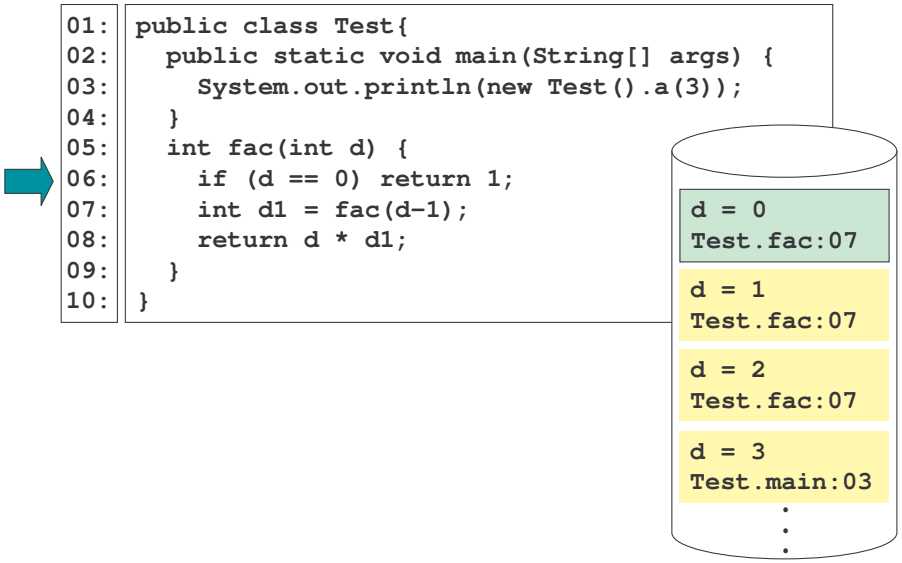
jvm ใช้ stack จัดการ recursive calls



jvm ใช้ stack จัดการ recursive calls

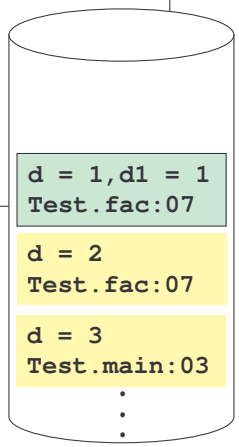


jvm ใช้ stack จัดการ recursive calls



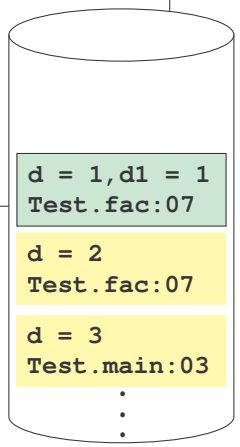
jvm ใช้ stack จัดการ recursive calls

```
01: public class Test{
02:   public static void main(String[] args) {
03:     System.out.println(new Test().a(3));
04:   }
05:   int fac(int d) {
06:     if (d == 0) return 1;
07:     int d1 = fac(d-1);
08:     return d * d1;
09:   }
10: }
```



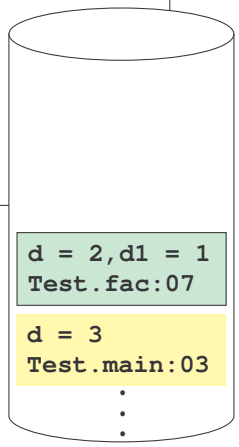
jvm ใช้ stack จัดการ recursive calls

```
01: public class Test{
02:   public static void main(String[] args) {
03:     System.out.println(new Test().a(3));
04:   }
05:   int fac(int d) {
06:     if (d == 0) return 1;
07:     int d1 = fac(d-1);
08:     return d * d1;
09:   }
10: }
```



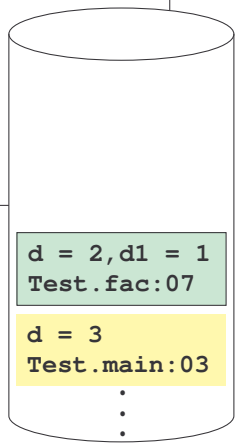
jvm ใช้ stack จัดการ recursive calls

```
01: public class Test{
02:   public static void main(String[] args) {
03:     System.out.println(new Test().a(3));
04:   }
05:   int fac(int d) {
06:     if (d == 0) return 1;
07:     int d1 = fac(d-1);
08:     return d * d1;
09:   }
10: }
```

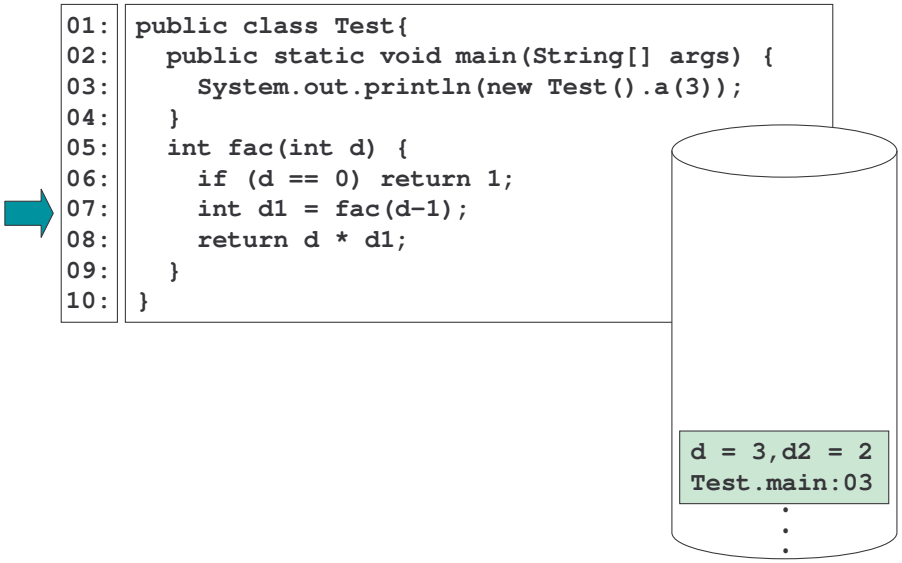


jvm ใช้ stack จัดการ recursive calls

```
01: public class Test{
02:   public static void main(String[] args) {
03:     System.out.println(new Test().a(3));
04:   }
05:   int fac(int d) {
06:     if (d == 0) return 1;
07:     int d1 = fac(d-1);
08:     return d * d1;
09:   }
10: }
```



jvm ใช้ stack จัดการ recursive calls



jvm ใช้ stack จัดการ recursive calls