2110101 : การทำโปรแกรมคอมพิวเตอร์

Methods

ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย

หัวข้อ

- บทบาทของ methods
- Methods ต่างๆ ของ class Math
- องค์ประกอบของ method
- การเรียกใช้
- Overloading

การเขียนโปรแกรม

- ปัญหาของโปรแกรมยาวๆ
  - อ่านยาก
  - debug ยุ่ง
  - แก้ไขปรับปรุงลำบาก
- ข้อสังเกตของโปรแกรมทั่วไป
  - สามารถแบ่งการทำงาน ออกเป็นส่วนๆ ตามหน้าที่
  - มีการทำงานซ้ำๆ บ่อยๆ

```
class Human {
   void main(...) {
   }
}
```

การเขียนโปรแกรม

- ทางแก้
  - แบ่งโปรแกรมเป็นส่วนย่อยๆ
  - แยกส่วนของโปรแกรมที่ซ้ำๆ คล้ายๆ กันออกมา
- เรียกส่วนย่อยเหล่านี้ว่า subroutine, subprogram, หรือ method
- แต่ละ method มีหน้าที่ในตัวเองอย่างเด่นชัด

```
class Human {
   main(...) {
     born();
     grown();
     sick();
     dead();
   }
   born(...) {
   }
   grown(...) {
   }
   sick(...) {
   }
   dead(...) {
   }
}
```

เมื่อต้องเขียนโปรแกรมใหม่

- คิดถึงโปรแกรมเก่าที่เคยเขียนไว้
- คิดถึงคลังคำสั่งของระบบที่มีให้ใช้ได้ (Java API)
- นำของที่มีอยู่มาแก้ไข (แฟ้ม .java)
- นำของที่มีอยู่มาใช้ใหม่ (แฟ้ม .class)
- เขียนคำสั่งเพื่อเชื่อมต่อของที่มีอยู่ให้ใช้ได้
- ถ้าจำเป็น จึงเขียนใหม่
  - เขียนให้อ่านง่าย (readability)
  - เขียนให้แก้ไขง่าย (maintainability)
  - เขียนให้สามารถนำไปใช้ในงานอื่นๆ ได้ในอนาคต (reuseability)

ตัวอย่าง method ที่เคยใช้

```
System.out.println("Hello World");
```

ส่ง "Hello World" เพื่อให้ System.out.println (แสดงออกทางจอภาพ)

```
int x = JLabIO.readInt("x = ");
```

ส่ง "x = " เพื่อให้ JLabIO.readInt (แสดงออกทางจอภาพ แล้วรอรับการป้อนจำนวนเต็มทางแป้นพิมพ) แล้วได้ผลลัพธ์เป็นจำนวนเต็มคืนมา เก็บไว้ในตัวแปร x

```
double r = Math.random();
```

ไม่ส่งอะไรให้ Math.random แต่จะได้ผลลัพธ์เป็นจำนวนจริงคืนมา (ซึ่งเป็นจำนวนจริงสุ่มในช่วง 0.0 ถึง 1.0) เก็บไว้ในตัวแปร x

Methods ต่างๆ ของคลาส Math

- abs(x) : absolute value
- acos(x) : arc cosine
- asin(x) : arc sine
- atan(x) : arc tangent
- cos(x) : trigonometric cosine
- sin(x) : trigonometric sine
- tan(x) : trigonometric tangent
- exp(x) : e^x
- log(x) : natural logarithm (base e)
- pow(x, y) : x^y

Methods ต่างๆ ของคลาส Math (ต่อ)

- ceil(x) : ceiling
- floor(x) : floor
- max(x, y) : the greater of x and y
- min(x , y) : the smaller of x and y
- random() : a random number in [0.0, 1.0)
- round(x) : the closest integer to x
- sqrt(x) : positive square root of x
- toDegrees(x) : convert angle in radian to degree
- toRadians(x) : convert angle in degree to radian

ตัวอย่าง Java API Help File

```
public static double sin(double a)
```

Returns the trigonometric sine of an angle. Special cases:

- If the argument is NaN or an infinity, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.

A result must be within 1 ulp of the correctly rounded result. Results must be semi-monotonic.

Parameters:

a - an angle, in radians.

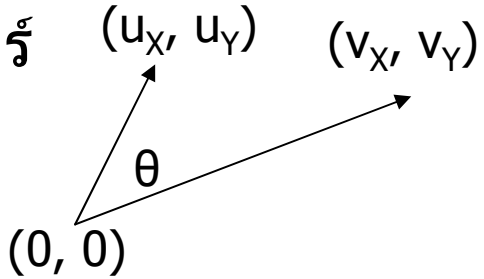
Returns:

the sine of the argument.

ulp - unit of the last place

โปรแกรมหามุมระหว่างสองเวกเตอร์

- Requirement
  - หามุม (เป็นองศา) ระหว่างสองเวกเตอร์
- Analysis
  - เวกเตอร์มีจุดเริ่มที่ (0, 0)
  - รับเวกเตอร์จากแป้นพิมพ์ โดยระบุตำแหน่งของจุดปลาย (u_X, u_Y) และ (v_X, v_Y)
  - สูตรการหามุมคือ



$$\theta = \cos^{-1}\left(\frac{u_X v_X + u_Y v_Y}{\left(\sqrt{u_X^2 + u_Y^2}\right)\left(\sqrt{v_X^2 + v_Y^2}\right)}\right)$$

  - แสดงมุมที่หาได้ทางจอภาพ

คล้ายกันมาก

แยกส่วนซ้าๆ คล้ายๆ กันเป็นเมท็อด

$$\cos(\theta) = \left(\frac{u_X v_X + u_Y v_Y}{\left(\sqrt{u_X^2 + u_Y^2}\right)\left(\sqrt{v_X^2 + v_Y^2}\right)} \right)$$

```
cosine = (ux * vx + uy * vy) /
         (Math.sqrt(ux * ux + uy * uy) *
          Math.sqrt(vx * vx + vy * vy));
```

```
cosine = (ux * vx + uy * vy) /
         (length(ux, uy) * length(vx, vy));
```

```
public static double length(double x, double y) {
  return Math.sqrt(x * x + y * y);
}
```

โปรแกรมหามุมระหว่างสองเวกเตอร์

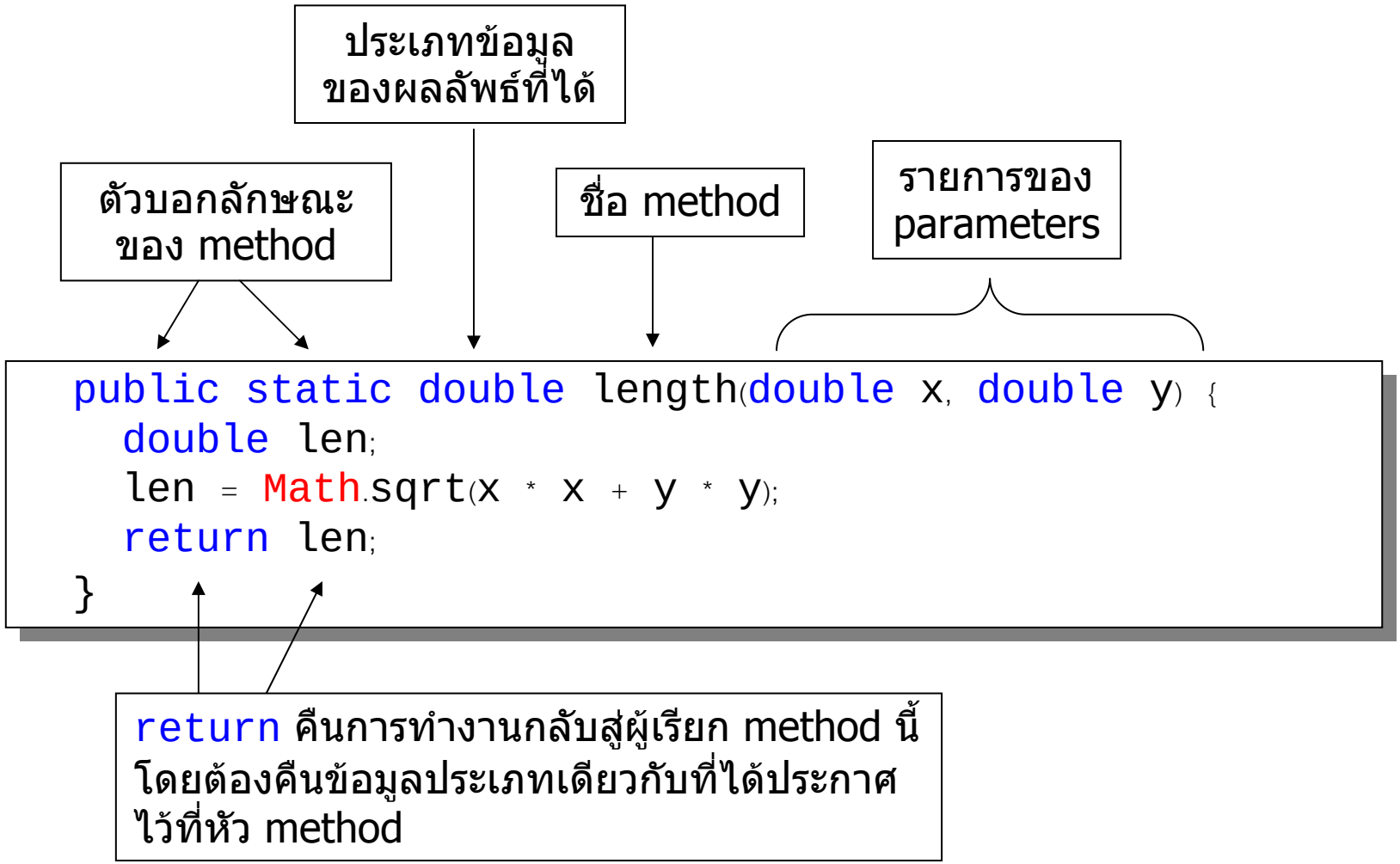
```
import jlab.JLabIO;
public class A {
  public static void main(String[] args) {
    double ux = JLabIO.readDouble("ux = ");
    double uy = JLabIO.readDouble("uy = ");
    double vx = JLabIO.readDouble("vx = ");
    double vy = JLabIO.readDouble("vy = ");

    double cosine = (ux * vx + uy * vy) /
                    (length(ux, uy) * length(vx, vy));

    double degree = Math.toDegrees(Math.acos(cosine));
    System.out.println(JLabIO.format(degree, 5, 2));
  }

  public static double length(double x, double y) {
    return Math.sqrt(x * x + y * y);
  }
}
```

องค์ประกอบของเมท็อด

- หัวเมท็อด
  - บอกลักษณะการใช้งานของเมท็อด
  - ระบุชื่อ ประเภทของข้อมูลที่รับ และประเภทของผลลัพธ์ที่ได้
- ตัวเมท็อด
  - ประกาศตัวแปรใช้ภายใน
  - บรรยายการทำงานของเมท็อด
  - คืนการทำงานสู่ผู้เรียก

```
public static double length(double x, double y) {
  double len;
  len = Math.sqrt(x * x + y * y);
  return len;
}
```

องค์ประกอบของเมท็อด



หัวเมท็อด : ประเภทข้อมูลของผลลัพธ์

- ประเภทข้อมูลพื้นฐาน (int, double, boolean, ...)
- ประเภทข้อมูลแบบคลาส (String, Point, ...)
- ถ้าเป็น method ที่ไม่คืนผลลัพธ์ ให้ใส่คำว่า void

```
public static int min(int a, int b)
```

```
int m = min(a, 0);
```

```
public static String toString(int i, int radix)
```

```
String s = toString(a, 16);
```

```
public static void gc()
```

```
gc();
```

หัวเมท็อด : ชื่อ

- การตั้งชื่อใช้กฎเหมือนกับตัวแปร
  - ประกอบด้วยตัวอักษร ตัวเลข ตัว $ หรือ _ ก็ได้
  - ห้ามขึ้นต้นด้วยตัวเลข
  - ไม่มีข้อจำกัดเรื่องความยาวของชื่อ
  - ตัวอักษรตัวใหญ่ไม่เหมือนตัวเล็ก
  - ต้องไม่ซ้ำกับคำสงวนของภาษาจาวา
- โดยทั่วไปชื่อเมท็อดขึ้นต้นด้วยตัวเล็ก
- มักตั้งชื่อเมท็อดให้เป็นกริยา

```
fillCircle  setEnabled  turnLeft  println  start  run
```

หัวเมท็อด : รายการของพารามิเตอร์

- รายการของ parameters
  - parameter เป็นตัวแปรสำหรับรับข้อมูลเข้ามาประมวลผล
  - แต่ละ parameter เขียนเหมือนการประกาศตัวแปร (แต่ไม่ต้องปิดท้ายด้วย semi-colon)
  - แต่ละ parameter คั่นด้วยจุลภาค (comma)
  - รายการของ parameters อยู่ภายในวงเล็บ (ถ้าไม่รับ parameter ใดๆ ก็ไม่ต้องใส่อะไรในวงเล็บ)

```
public static void turnLeft(double angle)
```

```
public static double pow(double a, double b)
```

```
public static String readString()
```

ตัวเมท็อด

```
public static double readDouble(String msg) {

  double x = 0;                          // local var. declaration
  boolean success = false;               // local var. declaration

  do {
    String s;                            // local var. declaration
    s = readString(msg);
    try {
      x = Double.parseDouble(s);
      success = true;
    } catch (NumberFormatException e) {
      System.err.println("invalid real number, try again");
      success = false;
    }
  } while (!success);

  return x;                              // return to caller

}
```

ตัวเมท็อด : การคืนการทำงานสู่ผู้เรียก

- เมื่อการทำงานพบคำสั่ง return ก็จะคืนการทำงานกลับไปสู่ผู้เรียก method
- สำหรับ method ที่ไม่มีการคืนผลลัพธ์
  - หลังคำว่า return ปิดท้ายด้วย ; เลย
  - ถ้าคำสั่งท้ายสุดของ method ไม่ใช่ return ก็จะเสมือนมี return หลังบรรทัดท้ายสุด

```
public static void printError(String msg) {
  if ( msg == null ) return;
  System.err.println( msg );
}
```

ตัวเมท็อด : การคืนการทำงานสู่ผู้เรียก

- สำหรับ method ที่มีการคืนผลลัพธ์
  - หลังคำว่า return ต้องตามด้วยนิพจน์
  - ค่าของนิพจน์ที่คืนเป็นผลลัพธ์ ต้องเป็นประเภทเดียวกับที่ประกาศไว้ที่หัว method หรือที่ระบบสามารถจะทำ promotion ให้ได้
  - คืนผลได้แค่ตัวเดียว

```
public static long clip(long a) {
  if ( a > 0 )
    return a;
  else
    return 0;      \\ 0 is promoted to 0L
}
```

ตัวเมท็อด : Local Variables

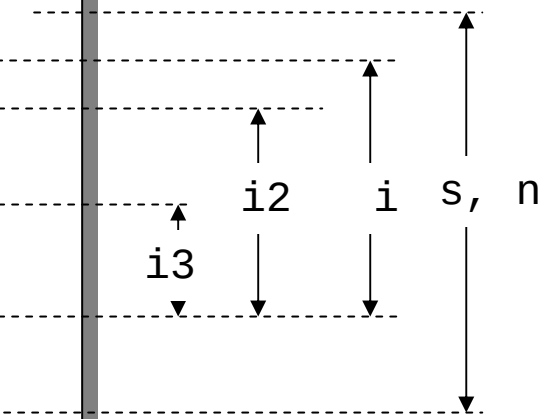
- Local variables คือตัวแปรที่ใช้ภายในเมท็อด
- ประกาศใน block ใด ใช้ได้เฉพาะใน block นั้น หรือ block ที่ซ้อนอยู่ชั้นในๆ ของ block นั้น
- ประกาศที่จุดใด ใช้ได้หลังจุดนั้นภายใน block

```
public static void test(int a) {
  int z = 7;
  { int x = z;  System.out.println( x );  }
  System.out.println( x );                    // wrong
  {
    System.out.println( x );                  // wrong
    int x = z;  System.out.println( x );
    { int y = z; System.out.println( y ); }
    { int x = z; System.out.println( x ); } // wrong
  }
}
```

ตัวเมท็อด : Local Variables

- ถูกสร้างใหม่ทุกครั้งที่ทำงานถึงจุดที่ประกาศ (ระบบสร้างให้โดยไม่ได้เติมค่าเริ่มต้น)
- ถูกทำลายหมดหลังจบ block ที่ถูกประกาศ
- ตัวแปรต่างๆ ในพารามิเตอร์ก็เป็น local variables

```
import jlab.JLabIO;
public class A {
  public static void main(String[] args) {
    int s = 0, n = JLabIO.readInt("n = ");
    for (int i = 0; i <= n; i++) {
      int i2 = i * i;
      s += i2;
      int i3 = i2 * i;
      s += i3;
    }
    System.out.println("s = " + s );
  }
}
```



สิ่งที่มักผิดบ่อย

```
public static printError(String msg) {
  System.err.println( msg );
}
```

```
public static int abs(int a) {
  if ( a < 0 ) a = -a;
}
```

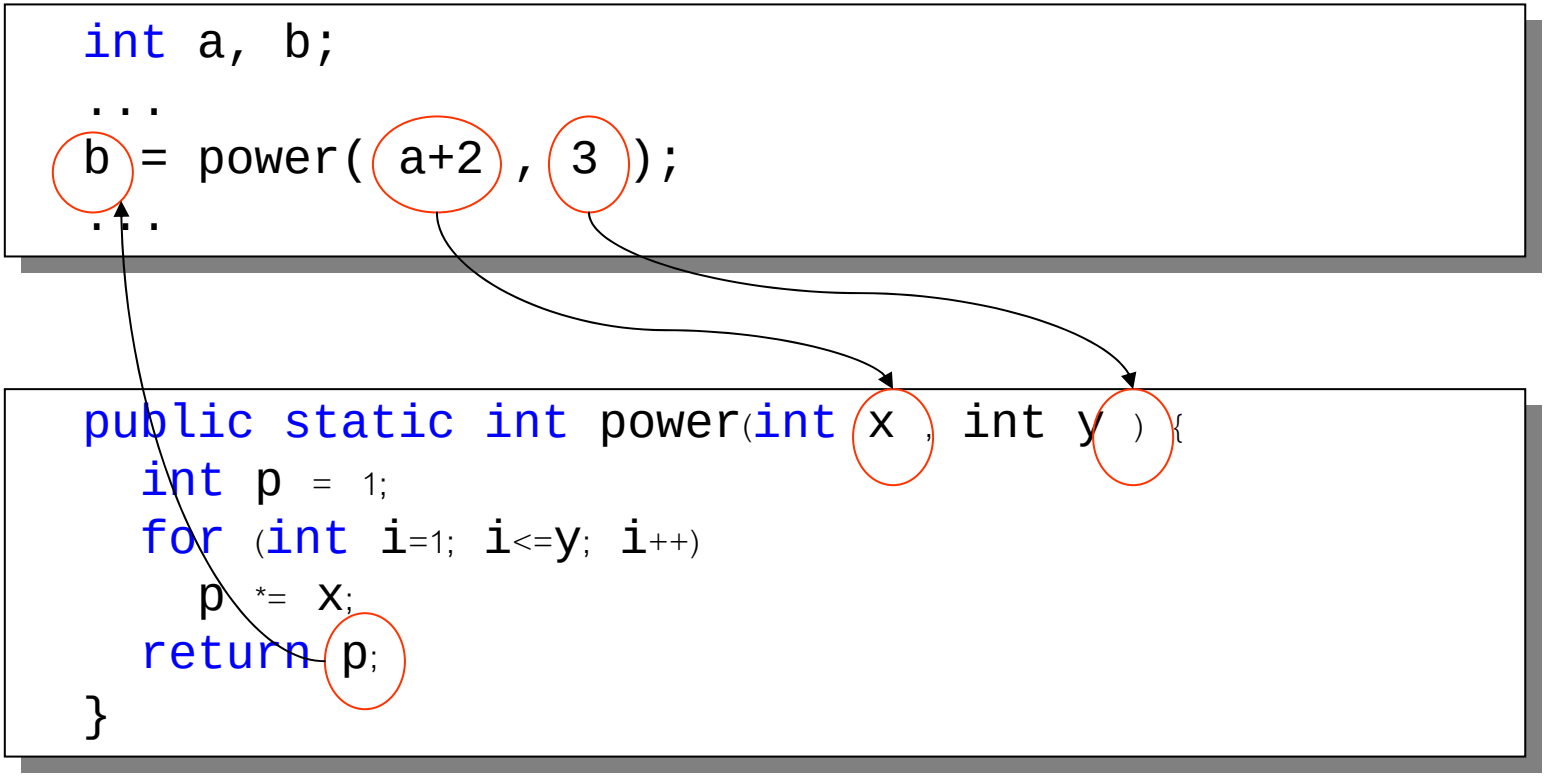
```
public static float max(float x, y) {
  if ( x > y ) return x; else return y;
}
```

```
public static double length(double dx, double dy) {
  double dx = Math.abs(dx);
  return Math.sqrt( dx*dx + dy*dy );
}
```

```
public static float getBlackColor() {
  return 0.0;
}
```

การเรียกใช้เมท็อด

- การส่งข้อมูลไป (arguments)
- การรับผลที่ได้คืนมา (returned value)
- ไม่มีข้อมูลส่งไป หรือไม่มีผลคืนมา ก็ได้



การเรียกใช้เมท็อด : การส่งข้อมูล

- จำนวนข้อมูลที่ส่งต้องตรงกับที่ประกาศไว้ที่หัวเมท็อด
- สิ่งที่ส่งไปต้องเป็นประเภทที่ตัวแปรที่หัวเมท็อดรับได้ (ถ้าจำเป็น ระบบจะทำ argument promotion ให้)

```
public static float max(float x, float y) {
  . . .
}
```

ถูก

```
x = max( (float) (a/3.0), 1f );
y = max( Float.parseFloat(str), 1 ); // 1 -> 1.0f
```

ผิด

```
x = max( (float) (a/2) ); // wrong number of args
y = max( "1.0", 1 ); // can't promote "1.0" to 1f
```

การเรียกใช้เมท็อด : ผลที่คืนกลับ

- ในกรณีที่เมท็อดมีการคืนค่า ก็ต้องมีตัวแปรซึ่งมีประเภทที่รับค่าของข้อมูลที่คืนมาได้

```
public static float max(float x, float y) {
   . . .
}
```

ถูก

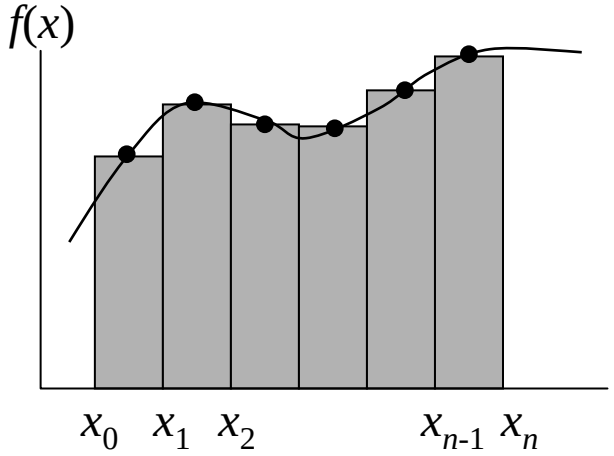
```
double x = max( 1.0f, (int) (x/3) );
int y = (int) max( 1, 2 );
float z = max( max(x, y), -10.5f ) );
```

ผิด

```
long x = max( 4.0f, 2.3f ); // float to long
float y = max( 4.0, 1.0 ); // double to float
max( 2.0f, 1.0f );  // max returns float
```

ตัวอย่าง : Integration

- แบ่งพื้นที่ใต้กราฟเป็นแถบๆ
- หาพื้นที่ของแต่ละแถบ แล้วหาผลรวม
- ใช้ midpoint rule ประมาณพื้นที่ใต้กราฟดังนี้



$$h = \frac{b-a}{n}$$

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} hf\left(\frac{x_{i-1}+x_i}{2}\right)$$

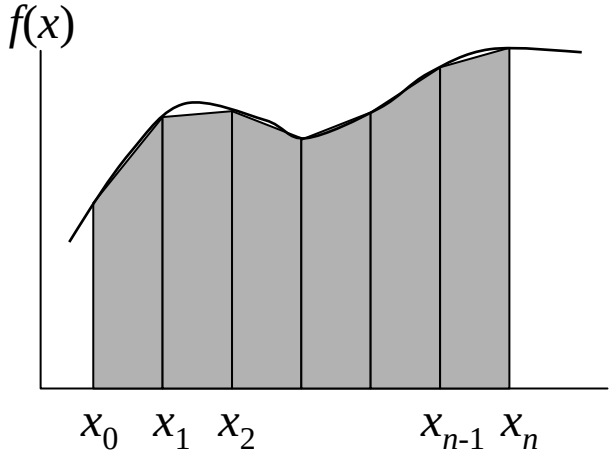
Integration using Midpoint Rule

```
public class Integration {
  public static void main(String[] args) {
    System.out.println(simpson(0, 10, 1000));
  }
  public static double midpoint(double a, double b, int n) {
    double h, s1, x;
    h = (b - a) / n;
    s1 = 0.0;
    x = a + h / 2.0;
    for (int i = 1; i <= n; i++) {
      s1 += h*f(x);
      x += h;
    }
    return s1;
  }
  public static double f(double x) {
    return 2 * x * x;
  }
}
```

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} hf\left(\frac{x_{i-1} + x_i}{2}\right)$$

ตัวอย่าง : Integration

- แบ่งพื้นที่ใต้กราฟเป็นแถบๆ
- หาพื้นที่ของแต่ละแถบ แล้วหาผลรวม
- ใช้ Simpson's rule ประมาณพื้นที่ใต้กราฟดังนี้



$$h = \frac{b-a}{n}$$

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f(a) + 4\left(\sum_{i=1,3,5,\ldots}^{n-1} f(a+ih) \right) + 2\left(\sum_{i=2,4,6,\ldots}^{n-2} f(a+ih) \right) + f(b) \right)$$

Integration using Simpson's Rule

```
public class Integration {
  public static void main(String[] args) {
    System.out.println(simpson(0, 10, 1000));
  }
  public static double simpson(double a, double b, int n) {
    double h, s1, s2, x;
    h = (b - a) / n;
    s1 = s2 = 0.0;
    x = a;
    for (int i = 1; i <= n; i++) {
      x += h;
      if (i % 2 == 1) // is odd ?
        s1 += f(x);
      else
        s2 += f(x);
    }
    return (h / 3 * (f(a) + 4 * s1 + 2 * s2 + f(b)));
  }
  public static double f(double x) {
    return 2 * x * x;
  }
}
```

$$\frac{h}{3}\left(f(a)+4\left(\sum_{i=1,3,5,\ldots}^{n-1} f(a+ih)\right)+2\left(\sum_{i=2,4,6,\ldots}^{n-2} f(a+ih)\right)+f(b)\right)$$

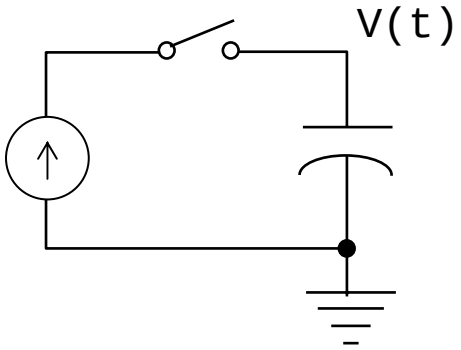
ตัวอย่าง : ความต่างศักย์ของตัวเก็บประจุ

- t=0 : ตัวเก็บประจุมีประจุเป็นศูนย์
- t=1 : ปิดสวิทซ์ แหล่งจ่ายประแส จ่าย

$$I(t) = 4(1 - e^{-0.5})e^{-0.5(t-1)}(1 - e^{-t})$$

- ประจุเพิ่ม

$$Q(t) = \int I(t)dt$$



- ความต่างศักย์เพิ่ม

$$V(t) = \frac{Q(t)}{C}$$

Voltage Across a Capacitor

```
public class CapacitorVoltage {
  public static void main(String[] args) {
    double c = 0.05;              // capacitance
    double v, q;                  // voltage and charge
    double a = 1.0, b = 10.0; // 1 to 10 seconds

    for (double t = a; t <= b; t++) {
      q = simpson(a, t, 1000);
      v = q / c;
      System.out.println(t + "\t" + q + "\t" + v);
    }
  }
  public static double simpson(double a, double b, int n) {
    ...
  }
  public static double f(double x) {
    return 4 * (1 - Math.exp(-0.5))
             * Math.exp(-0.5 * (t - 1))
             * (1 - Math.exp(-t));
  }
}
```

Method Overloading

- ชื่อเมท็อดซ้ำได้ ถ้ารายการพามิเตอร์ไม่เหมือนกัน
- ระบบจะเรียกเมท็อดที่เหมาะสมให้

```
public static int max(int x, int y) {
  if ( x > y ) return x; else return y;
}
public static double max(double x, double y) {
  if ( x > y ) return x; else return y;
}
public static double max(double x, double y, double z) {
  if ( x > y && x > z ) return x;
  else if ( y > x && y > z ) return y;
  else return z;
}
```

Method Overloading

- ไม่สามารถ overload เมท็อดที่รายการพามิเตอร์เหมือน แต่ประเภทข้อมูลที่คืนไม่เหมือน
- ความเหมือนไม่เหมือนของพารามิเตอร์ อยู่ที่ประเภทข้อมูล ไม่เกี่ยวกับชื่อที่ใช้

```
public static int max(int x, int y) {
  if ( x > y ) return x; else return y;
}

public static double max(int x, int y) {
  if ( x > y ) return (double) x; else return (double) y;
}

public static int max(int a, int b) {
  if ( x > y ) return a; else return b;
}
```