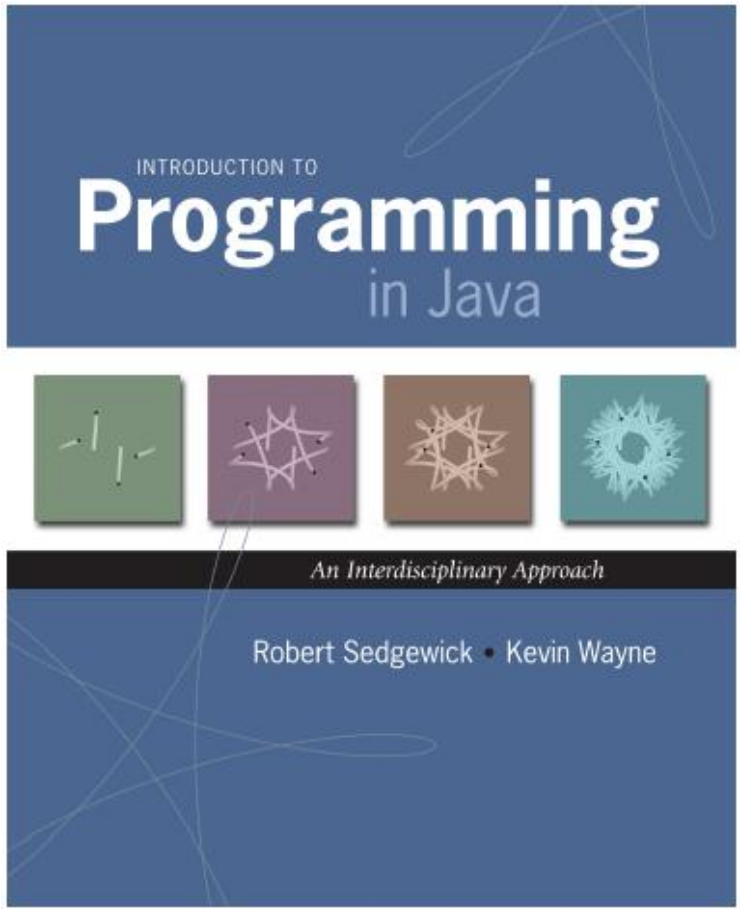
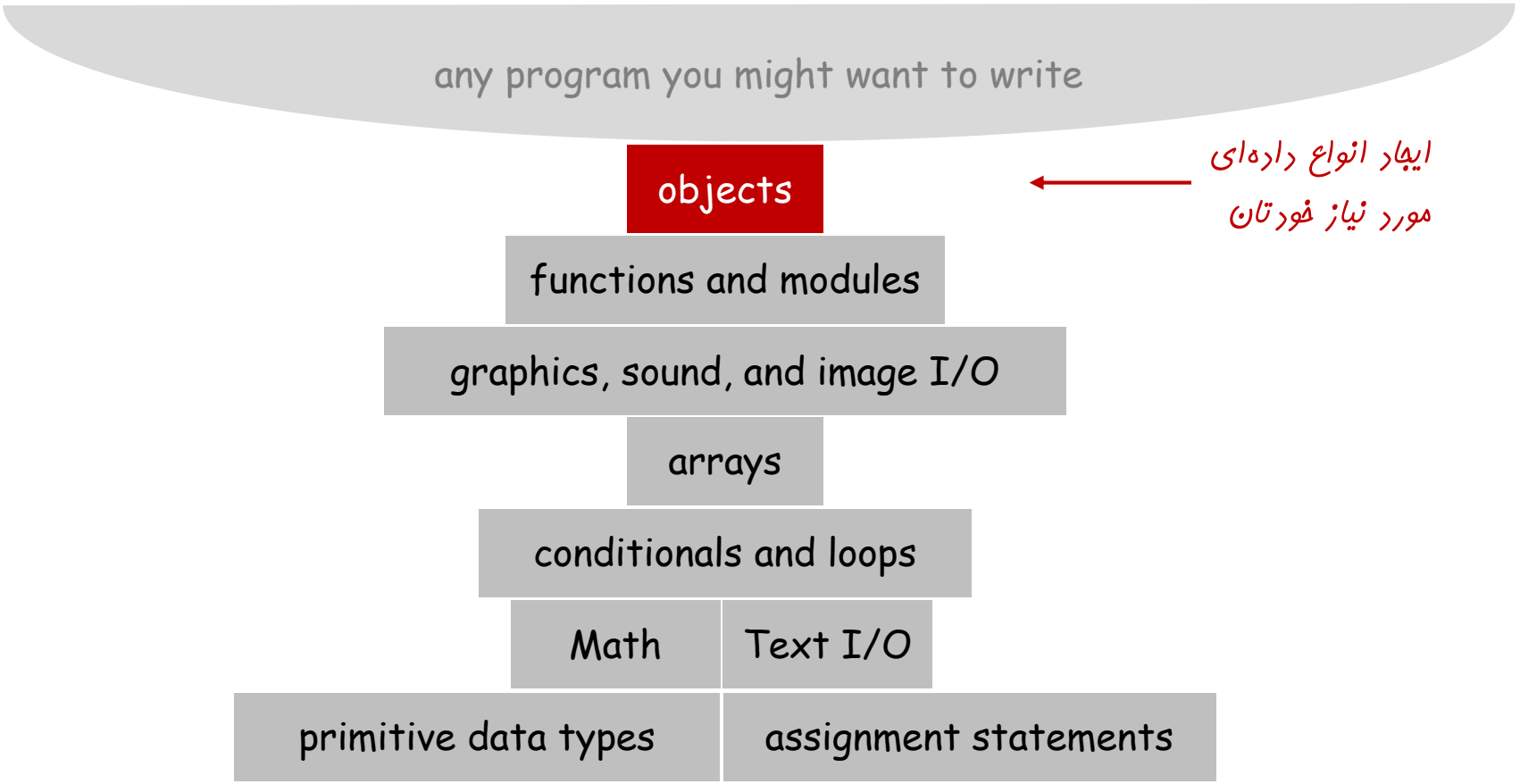
برنامه‌نویسی شی‌گرا:
ایجاد انواع داده‌ای جدید

سید ناصر رضوی **www.snrazavi.ir**
۱۳۹۶

۳-۲ ایجاد انواع داده‌ای



اجزای برنامه‌نویسی



انواع داده‌ای

- نوع داده‌ای. یک مجموعه از مقادیر و عملیات قابل انجام بر روی آنها.
- انواع اولیه.

Data Type	*Values*	*Operations*
boolean	true, false	not, and, or, xor
int	-2^{31} to $2^{31} - 1$	add, subtract, multiply
double	2^{64} possible real values	add, subtract, multiply

- بخش قبل. نوشتن برنامه به منظور استفاده از انواع داده‌ای موجود.
- این بخش. نوشتن برنامه به منظور ایجاد انواع داده‌ای جدید مورد نیاز.

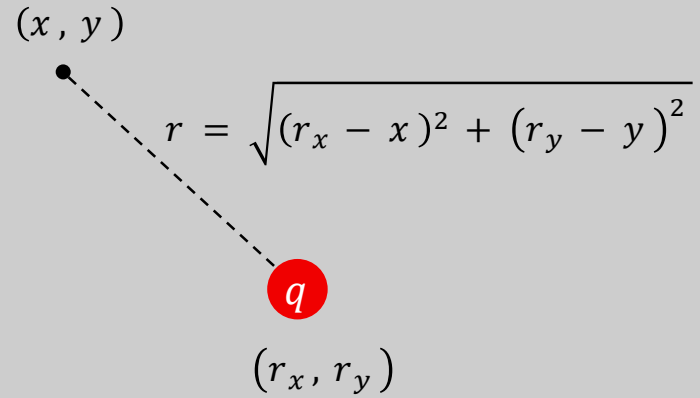
تعریف انواع داده‌ای در جاوا

- برای تعریف یک نوع داده‌ای باید موارد زیر را مشخص کنیم.
  - مجموعه مقادیر.
  - عملیات قابل انجام بر روی این مجموعه از مقادیر.

- کلاس جاوا. تعریف کننده یک نوع داده‌ای با مشخص کردن:
  - متغیرهای نمونه. (مجموعه مقادیر)
  - متدها. (عملیات قابل انجام بر روی این مجموعه از مقادیر)
  - سازنده‌ها. (ایجاد و آماده‌سازی اشیای جدید)

نوع داده‌ای ذرات باردار

- هدف. ایجاد یک نوع داده‌ای به منظور کار کردن با ذرات باردار.
- مجموعه مقادیر. سه مقدار حقیقی. [مکان ذره و مقدار بار الکتریکی]
- عملیات.
  - ایجاد یک ذره باردار جدید در مکان (r_x,r_y) با بار الکتریکی q.
  - تعیین پتانسیل الکتریکی V ناشی از این ذره در مکان (x,y).
  - تبدیل به رشته.



$$V = k\frac{q}{r}$$

$k = 8.99 \times 10^9 \ \text{N.m}^2/\text{C}^2$

نوع داده‌ای ذرات باردار

- هدف. ایجاد یک نوع داده‌ای به منظور کار کردن با ذرات باردار.
- مجموعه مقادیر. سه مقدار حقیقی. [مکان ذره و مقدار بار الکتریکی]
- API.

```
public class Charge
          Charge(double rx, double ry, double q)
  double  potentialAt(double x, double y)
  String  toString()
```

پتانسیل القایی در نقطه (x, y)

نمایش رشته‌ای

نوع داده‌ای ذرات باردار: یک برنامه مشتری

- برنامه مشتری.
  - از عملیات تعریف شده بر روی نوع داده‌ای به منظور محاسبه کمیت مورد نظر استفاده می‌کند.

```
public static void main(String[] args) {
   double x = Double.parseDouble(args[0]);
   double y = Double.parseDouble(args[1]);
   Charge c1 = new Charge(.51, .63, 21.3);
   Charge c2 = new Charge(.13, .94, 81.9);
   double v1 = c1.potentialAt(x, y);
   double v2 = c2.potentialAt(x, y);
   StdOut.println(c1);
   StdOut.println(c2);
   StdOut.println(v1 + v2);
}
```

تبدیل خودکار به نوع رشته‌ای

```
% java Charge .50 .50
21.3 at (0.51, 0.63)
81.9 at (0.13, 0.94)
2.7493690708591 2e12
```

آناتومی متغیرهای نمونه

- متغیرهای نمونه. مشخص کننده مجموعه مقادیر.
  - متغیرهای نمونه را بیرون از متدها تعریف کنید.
  - همیشه قابلیت دسترسی آنها را به صورت **private** تعریف کنید.
  - اگر مقدار یک متغیر نمونه هرگز تغییر نمی‌کند، از اصلاح‌گر **final** استفاده کنید.

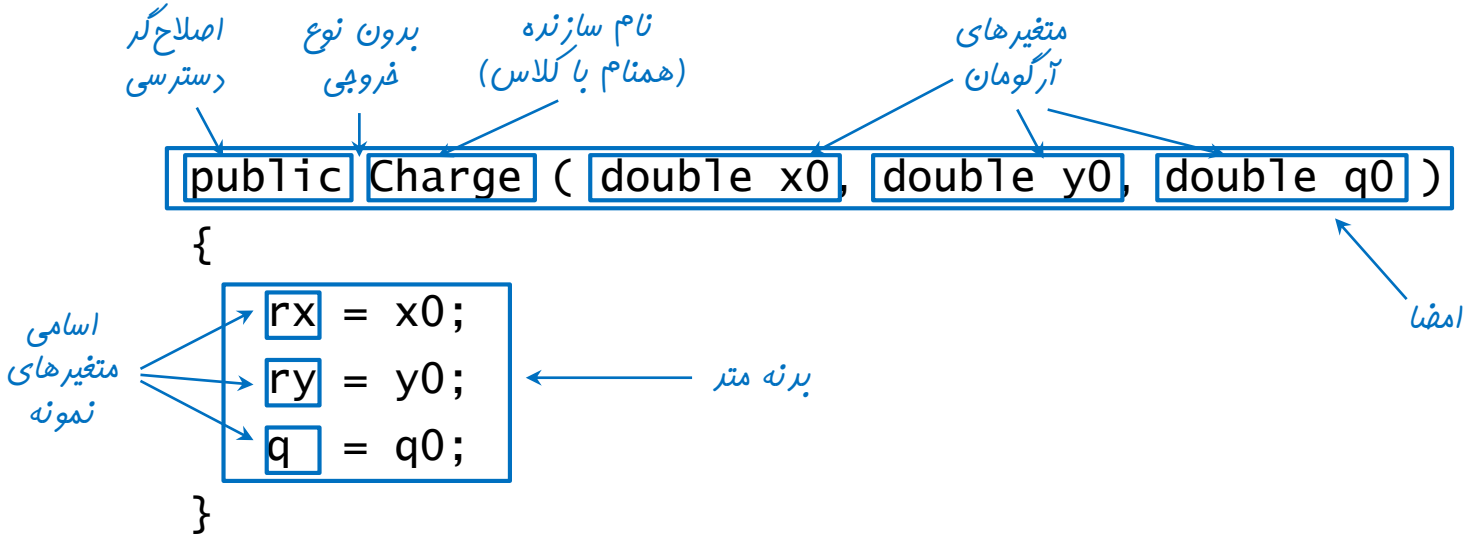
```
public class Charge
{
   private final double rx, ry;
   private final double q;
   .
   .
   .
}
```

اعلان متغیرهای نمونه

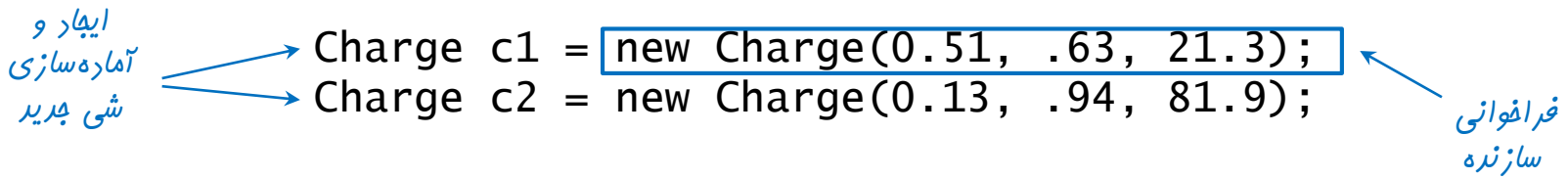
اصلاح‌گرها

آناتومی سازنده‌ها

- سازنده. ایجاد و آماده‌سازی اشیای جدید.

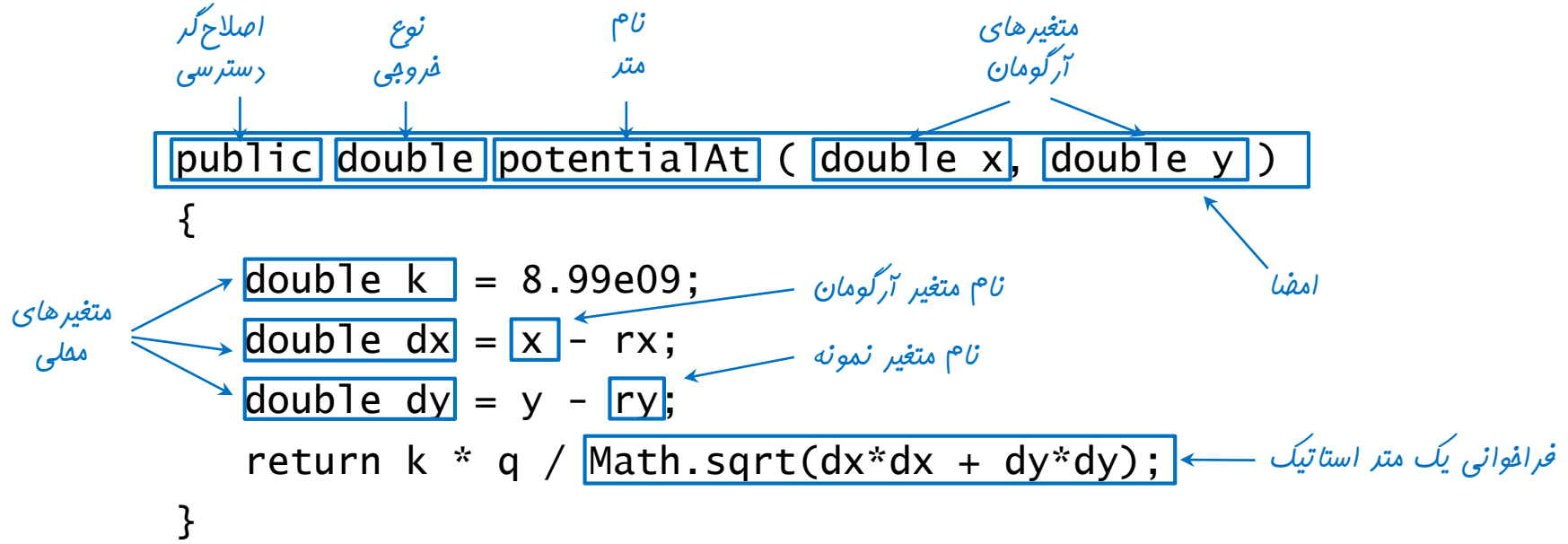


- فراخوانی یک سازنده. از عملگر **new** برای ایجاد یک شی جدید استفاده کنید.

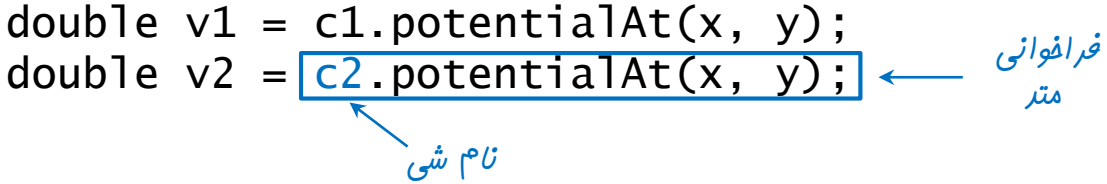


آناتومی متدهای نمونه

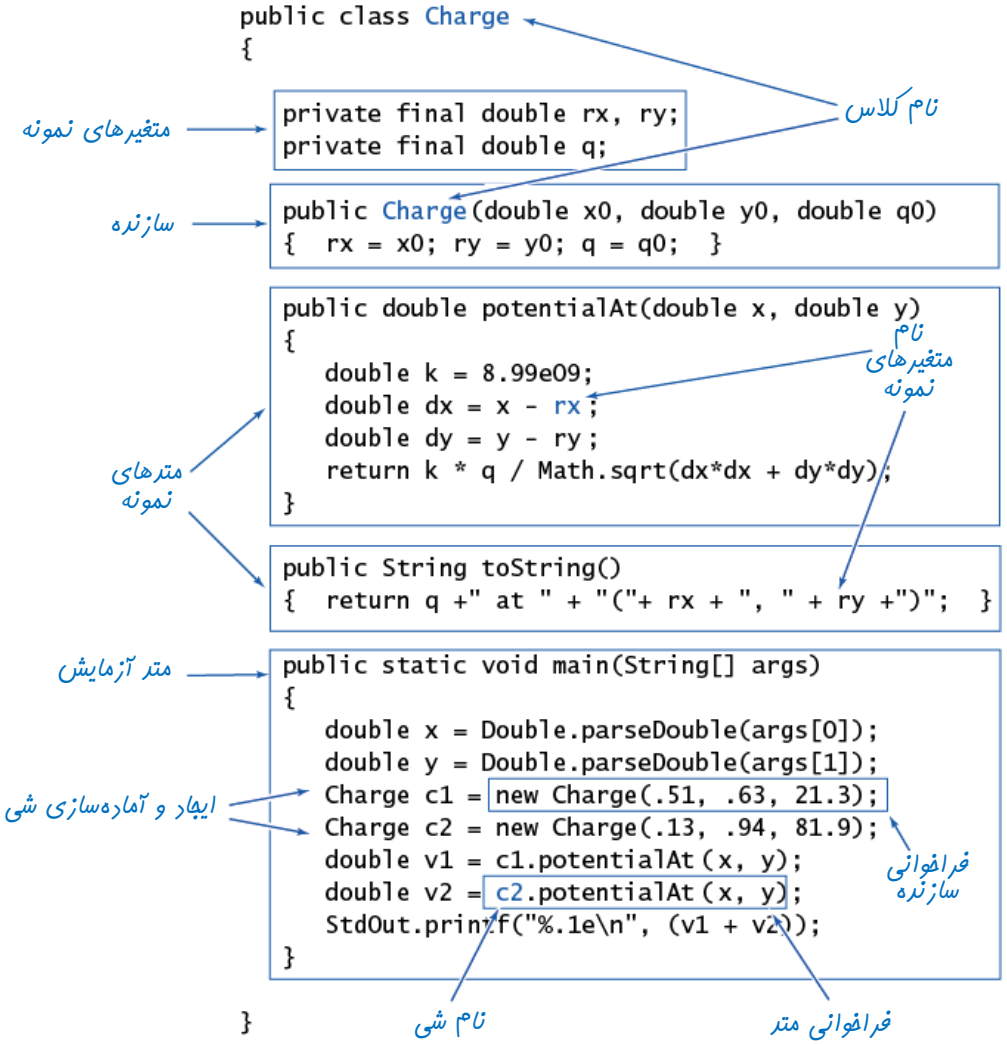
- متد. تعریف کننده عملیات قابل انجام بر روی متغیرهای نمونه.



- فراخوانی یک متد. از عملگر نقطه برای فراخوانی یک متد استفاده کنید.



آناتومی یک کلاس

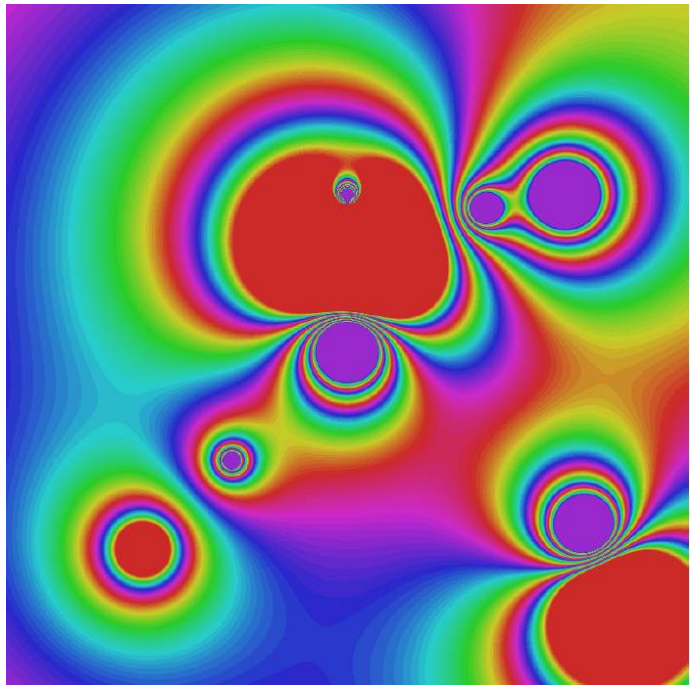


به تصویر کشیدن پتانسیل

- به تصویر کشیدن پتانسیل. از ورودی استاندارد مشخصات N بار الکتریکی را بخوانید؛ سپس پتانسیل کل هر نقطه را در یک مربع واحد محاسبه کنید.

```
% more charges.txt
9
.51 .63 -100
.50 .50 40
.50 .72 10
.33 .33 5
.20 .20 -10
.70 .70 10
.82 .72 20
.85 .23 30
.90 .12 -50
```

```
% java Potential < charges.txt
```



به تصویر کشیدن پتانسیل

- آرایه‌ای از اشیا.
  - با استفاده از دستور **new** به آرایه حافظه تخصیص دهید؛
  - سپس با استفاده از دستور **new** به هر یک از اشیاء موجود در آن حافظه تخصیص دهید.

```
// read in the data
int N = StdIn.readInt();
Charge[] a = new Charge[N];
for (int i = 0; i < N; i++) {
   double x0 = StdIn.readDouble();
   double y0 = StdIn.readDouble();
   double q0 = StdIn.readDouble();
   a[i] = new Charge(x0, y0, q0);
}
```

```
% more charges.txt
9
.51 .63 -100
.50 .50 40
.50 .72 10
.33 .33 5
.20 .20 -10
.70 .70 10
.82 .72 20
.85 .23 30
.90 .12 -50
```

به تصویر کشیدن پتانسیل


```
// plot the data
int SIZE = 512;
Picture pic = new Picture(SIZE, SIZE);
for (int i = 0; i < SIZE; i++) {
   for (int j = 0; j < SIZE; j++) {
      double V = 0.0;
      for (int k = 0; k < N; k++) {
         double x = 1.0 * i / SIZE;
         double y = 1.0 * j / SIZE;
         V += a[k].potentialAt(x, y);
      }
      Color color = getColor(V);
      pic.set(i, SIZE-1-j, color);
   }
}
pic.show();
```

محاسبه رنگ به عنوان
تابعی از مقدار پتانسیل

در نظر گرفتن گوشه بالا سمت
چپ به عنوان مبدأ مختصات

$$V = \sum_k (k\, q_k / r_k)$$

گرافیک لاک‌پشت

- هدف. ایجاد یک نوع داده‌ای به منظور کنترل حرکت یک لاک‌پشت در صفحه.
- مجموعه مقادیر. موقعیت و جهت لاک‌پشت.

```
public class Turtle
```

`Turtle(double x0, double y0, double a0)`	ایجاد یک لاک‌پشت جدید در مکان (x0, y0) با زاویه a0 درجه با محور x
`void turnLeft(double delta)`	به اندازه دلتا درجه به چپ گردش کن
`void goForward(double step)`	به اندازه step به جلو برو و یک خط ترسیم کن

```
Turtle turtle = new Turtle(0.0, 0.0, 0.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
```

ترسیم یک مربع

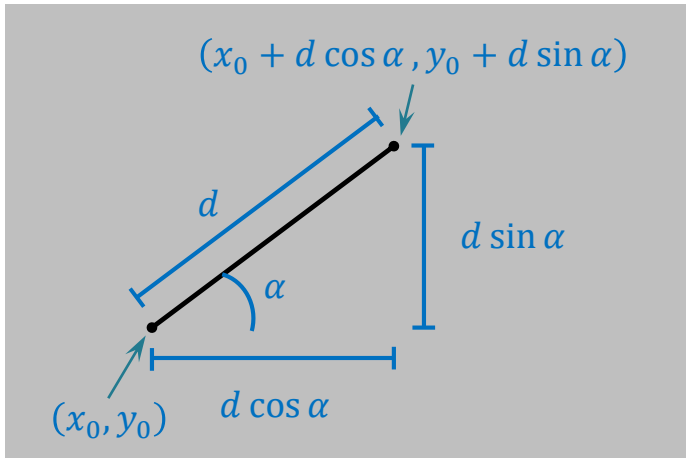
گرافیک لاک‌پشت


```
public class Turtle {

   private double x, y;      // turtle is at (x, y)
   private double angle;     // facing this direction

   public Turtle(double x0, double y0, double a0) {
         x = x0;
         y = y0;
         angle = a0;
   }

   public void turnLeft(double delta) {
         angle += delta;
   }

   public void goForward(double d) {
         double oldx = x;
         double oldy = y;
         x += d * Math.cos(Math.toRadians(angle));
         y += d * Math.sin(Math.toRadians(angle));
         StdDraw.line(oldx, oldy, x, y);
   }
}
```

سازنده

متر

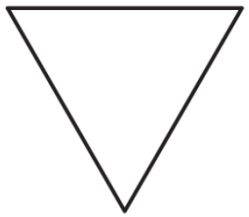
متر



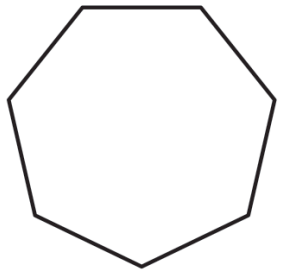
چندضلعی


```
public class Ngon {
   public static void main(String[] args) {
      int N         = Integer.parseInt(args[0]);
      double angle = 360.0 / N;
      double step  = Math.sin(Math.toRadians(angle/2.0));
      Turtle turtle = new Turtle(0.5, 0, angle/2.0);
      for (int i = 0; i < N; i++) {
         turtle.goForward(step);
         turtle.turnLeft(angle);
      }
   }
}
```

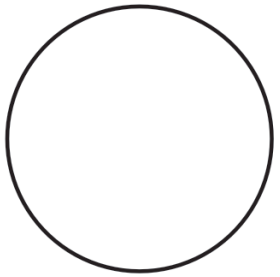
% java Turtle 3



% java Turtle 7

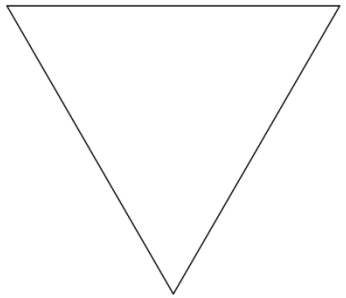


% java Turtle 30

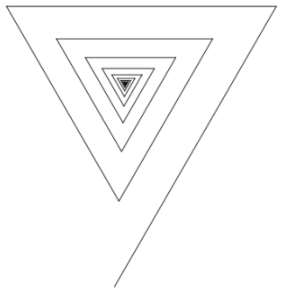


مارپیچ

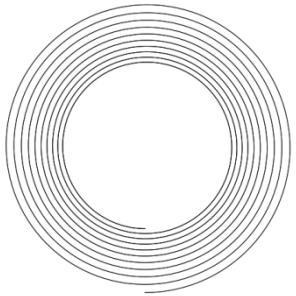

```
public class Spiral {
   public static void main(String[] args) {
      int N        = Integer.parseInt(args[0]);
      double decay = Double.parseDouble(args[1]);
      double angle = 360.0 / N;
      double step  = Math.sin(Math.toRadians(angle/2.0));
      Turtle turtle = new Turtle(0.5, 0, angle/2.0);
      for (int i = 0; i < 10 * N; i++) {
         step /= decay;
         turtle.goForward(step);
         turtle.turnLeft(angle);
      }
   }
}
```



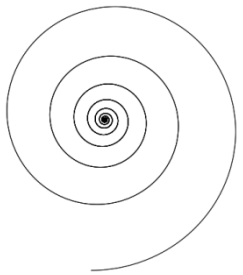
3 1.0



3 1.2

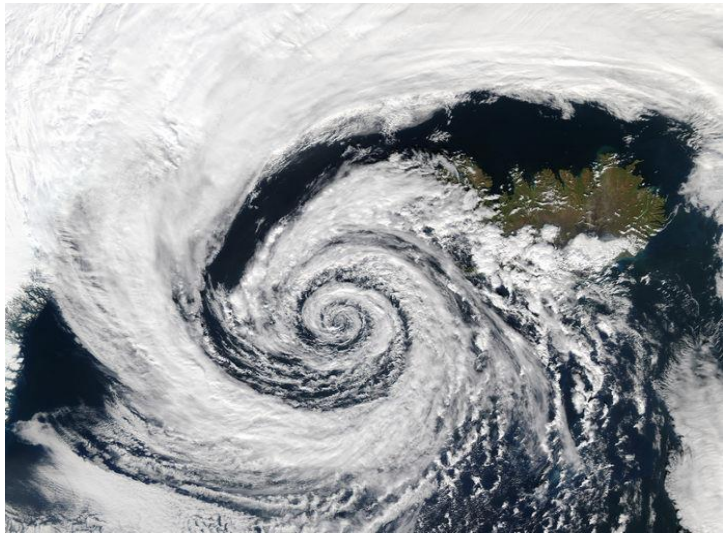


1440 1.00004



1440 1.0004

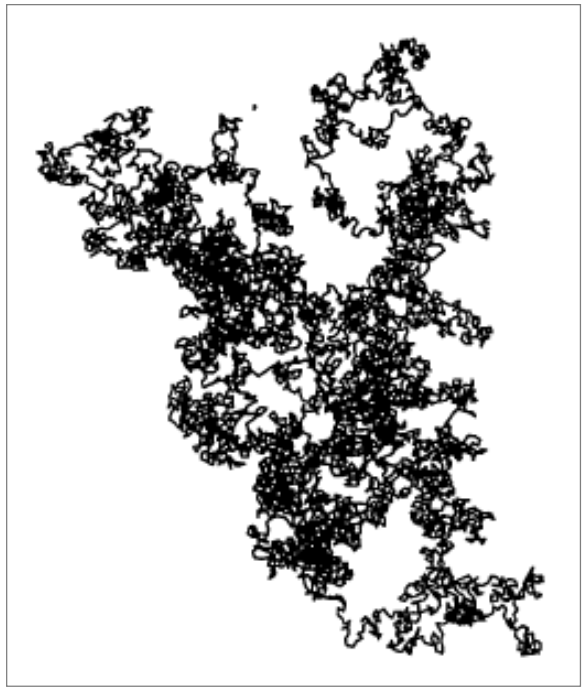
شکل‌های مارپیچ در طبیعت



حرکت تصادفی (براونی)


```
public class DrunkenTurtle {

    public static void main(String[] args) {
       int T = Integer.parseInt(args[0]);
       double step = Double.parseDouble(args[1]);

       Turtle turtle = new Turtle(0.5, 0.5, 0.0);

       for (int t = 0; t < T; t++) {
          turtle.turnLeft(360.0 * Math.random());
          turtle.goForward(step);
       }
    }
}
```

10000 .01



اعداد مختلط

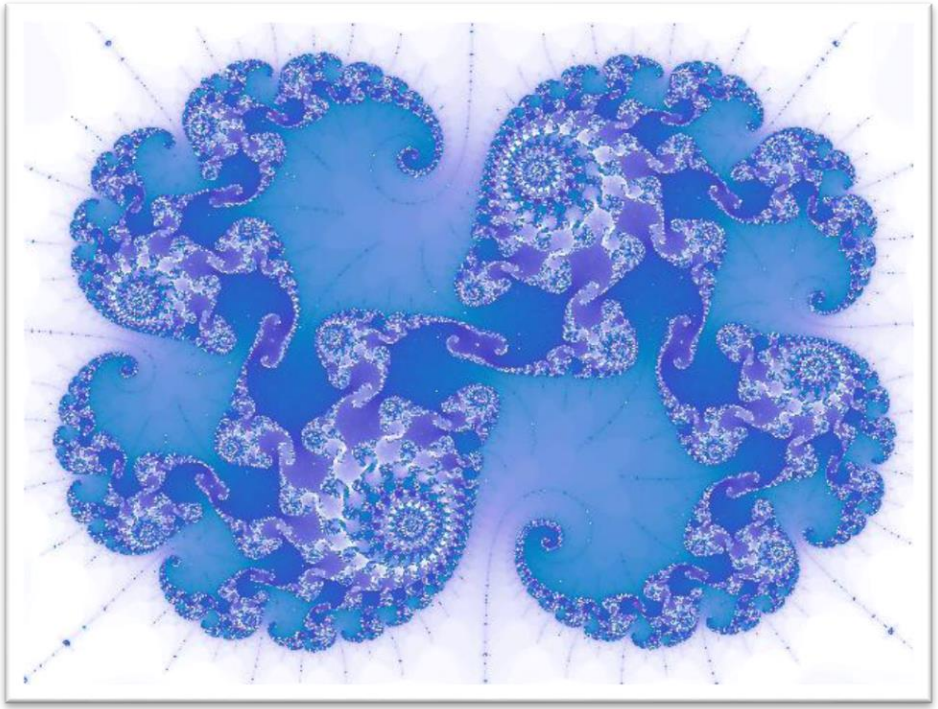
- هدف. ایجاد یک نوع داده‌ای به منظور کار کردن با اعداد مختلط.
- مجموعه مقادیر. دو عدد حقیقی: قسمت حقیقی و موهومی.

```
public class Complex
            Complex(double real, double imag)
 Complex    plus(Complex b)          جمع این عدد با b
 Complex    times(Complex b)         ضرب این عدد در b
  double    abs()                    اندازه
  double    re()                     بخش حقیقی
  double    im()                     بخش موهومی
  String    toString()               نمایش رشته‌ای
```

کاربردهای اعداد مختلط

- اهمیت. یک مدل ریاضی فراگیر.

- کاربردها.
  - برخال‌ها (فرکتال‌ها).
  - مقاومت ظاهری در مدارهای RLC.
  - پردازش سیگنال و تحلیل فوریه.
  - نظریه کنترل و تبدیلات لاپلاس.
  - مکانیک کوانتومی و فضاهای هیلبرت.
  - ...



نوع داده‌ای عدد مختلط: یک برنامه مشتری

- برنامه مشتری.
  - از عملیات تعریف شده بر روی نوع داده‌ای به منظور محاسبه کمیت مورد نظر استفاده می‌کند.

```
public static void main(String[] args) {
   Complex a = new Complex( 3.0, 4.0);
   Complex b = new Complex(-2.0, 3.0);
   Complex c = a.times(b);
   StdOut.println("a = " + a);
   StdOut.println("b = " + b);
   StdOut.println("c = " + c);
}
```

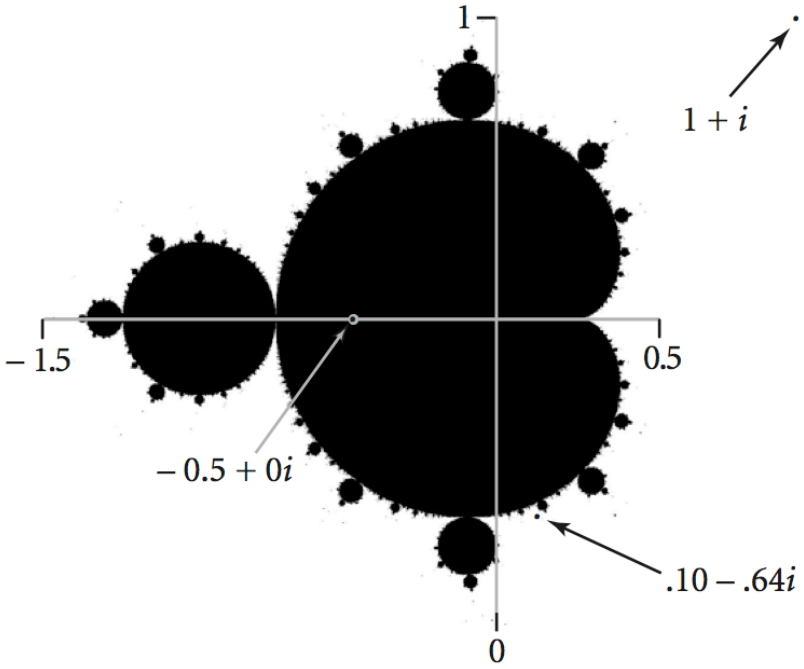
```
% java TestClient
a = 3.0 + 4.0i
b = -2.0 + 3.0i
c = -18.0 + 1.0i
```

نوع داده‌ای عدد مختلط: پیاده‌سازی


```
public class Complex {
    private final double re;
    private final double im;

    public Complex(double real, double imag) {
        re = real;
        im = imag;
    }

    public String toString() { return re + " + " + im + "i"; }

    public double abs() { return Math.sqrt(re*re + im*im); }

    public Complex plus(Complex b) {
        double real = re + b.re;
        double imag = im + b.im;
        return new Complex(real, imag);
    }

    public Complex times(Complex b) {
        double real = re * b.re - im * b.im;
        double imag = re * b.im + im * b.re;
        return new Complex(real, imag);
      }
}
```

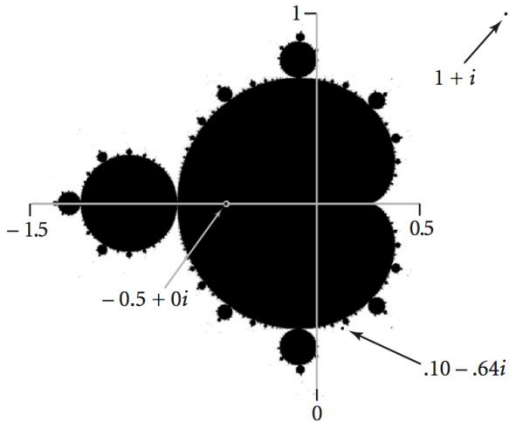
مجموعه مندلبرو

- مجموعه مندلبرو. یک مجموعه از اعداد مختلط.
- ترسیم. اگر `z = x + iy` به مجموعه تعلق دارد، نقطه `(x, y)` را به رنگ سیاه ترسیم کن.



- هیچ فرمول ساده‌ای برای تعیین این که کدام اعداد به مجموعه تعلق دارند، وجود ندارد.
- اما یک توصیف الگوریتمی وجود دارد!

مجموعه مندلبرو



- مجموعه مندلبرو. آیا عدد مختلط z_0 به مجموعه تعلق دارد؟
  - تکرار: $z_{t+1} = (z_t)^2 + z_0$
  - اگر $|z_t|$ به بی‌نهایت واگرا شود، آنگاه z_0 به مجموعه تعلق ندارد؛
  - در غیر این صورت، z_0 به مجموعه تعلق دارد.

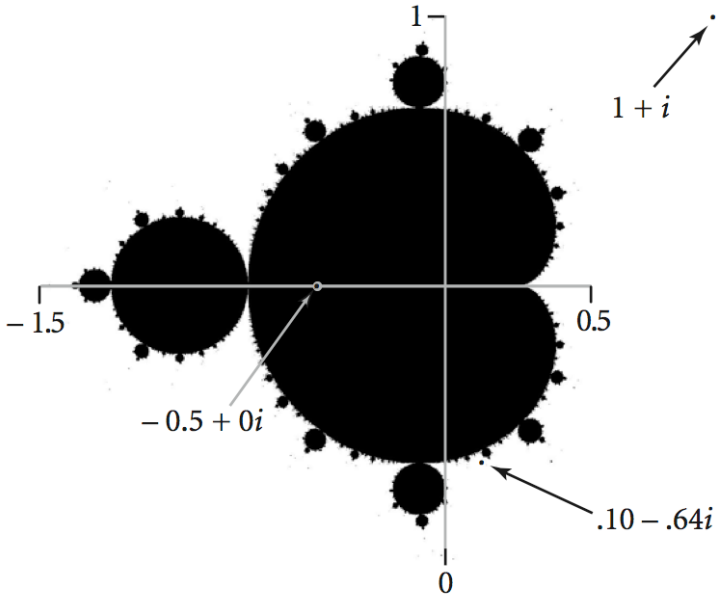
عدد 1 + i در مجموعه مندلبرو قرار ندارد

t	z_t
0	1 + i
1	1 + 3i
2	-7 + 7i
3	1 - 97i
4	-9407 - 193i
5	88454401 + 3631103i

عدد 1/2 در مجموعه مندلبرو قرار دارد

t	z_t
0	-1/2 + 0i
1	-1/4 + 0i
2	-7/16 + 0i
3	-79/256 + 0i
4	-26527/65536 + 0i
5	-1443801919/4294967296 + 0i

ترسیم مجموعه مندلبرو



- ملاحظات عملی.
  - نمی‌توان بی‌نهایت نقطه را ترسیم نمود.
  - نمی‌توان یک حلقه را بی‌نهایت بار تکرار کرد.

- یک راه‌حل تقریبی.
  - از یک شبکه N در N از نقاط صفحه نمونه‌برداری کن.
  - حقیقت: اگر $|z_t| > 2$ به ازای هر t، آنگاه z در مجموعه مندلبرو قرار ندارد.
  - شبه حقیقت: اگر $|z_{255}| \leq 2$، آنگاه احتمالاً z در مجموعه مندلبرو قرار دارد.

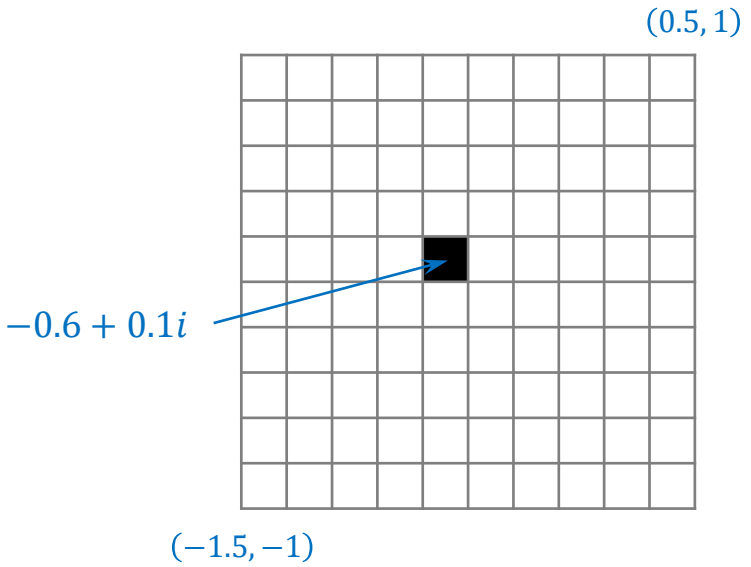
ترسیم مجموعه مندلبرو

- ملاحظات عملی.
  - نمی‌توان بی‌نهایت نقطه را ترسیم نمود.
  - نمی‌توان یک حلقه را بی‌نهایت بار تکرار کرد.

- یک راه‌حل تقریبی.
  - از یک شبکه N در N از نقاط صفحه نمونه‌برداری کن.
  - حقیقت: اگر $|z_t| > 2$ به ازای هر t، آنگاه z در مجموعه مندلبرو قرار ندارد.
  - شبه حقیقت: اگر $|z_{255}| \leq 2$، آنگاه احتمالاً z در مجموعه مندلبرو قرار دارد.

ترسیم مجموعه مندلبرو

- یک راه‌حل تقریبی.
  - از یک شبکه N در N از نقاط صفحه نمونه‌برداری کن.
  - حقیقت: اگر $|z_t| > 2$ به ازای هر t، آنگاه z در مجموعه مندلبرو قرار ندارد.
  - شبه حقیقت: اگر $|z_{255}| \leq 2$، آنگاه احتمالاً z در مجموعه مندلبرو قرار دارد.



نوع داده‌ای عدد مختلط: یک برنامه مشتری دیگر

- تابع مندلبرو با اعداد مختلط.
  - آیا z_0 در مجموعه مندلبرو قرار دارد؟
  - خروجی:
    - سفید (قطعاً خیر)
    - سیاه (احتمالاً بله)

```
public static Color mand(Complex z0) {
   Complex z = z0;
   for (int t = 0; t < 255; t++) {
      if (z.abs() > 2.0) return StdDraw.WHITE;
      z = z.times(z).plus(z0);
   }
   return StdDraw.BLACK;
}
```

$z = z^2 + z_0$

- تصاویر جذاب‌تر. به جای رنگ سفید از سطوح خاکستری یا رنگی استفاده کن.

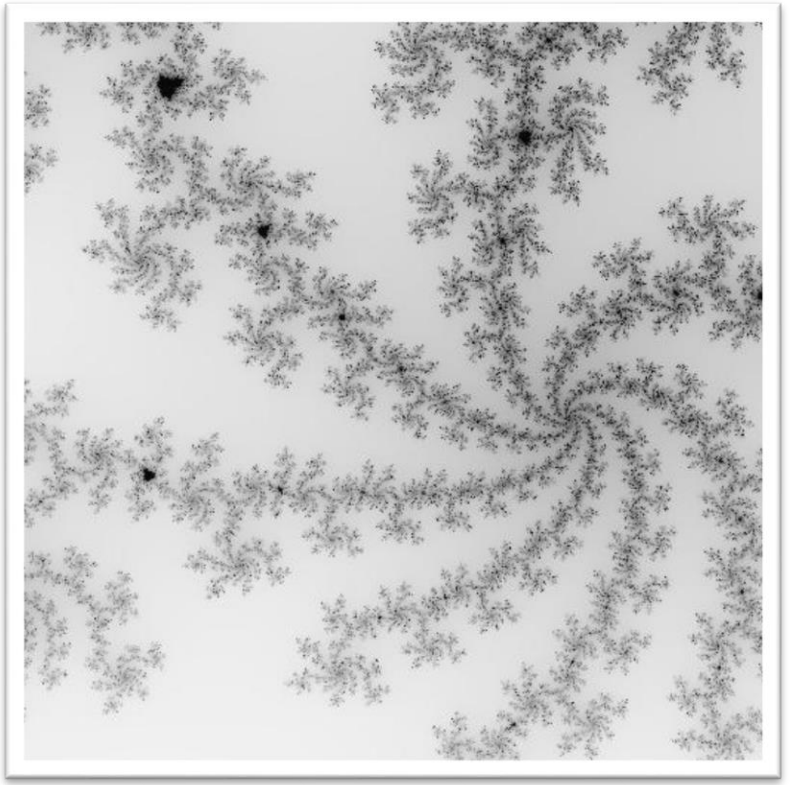
```
new Color(255-t, 255-t, 255-t)
```

نوع داده‌ای عدد مختلط: یک برنامه مشتری دیگر

- ترسیم مجموعه مندلبرو با استفاده از سطوح خاکستری.

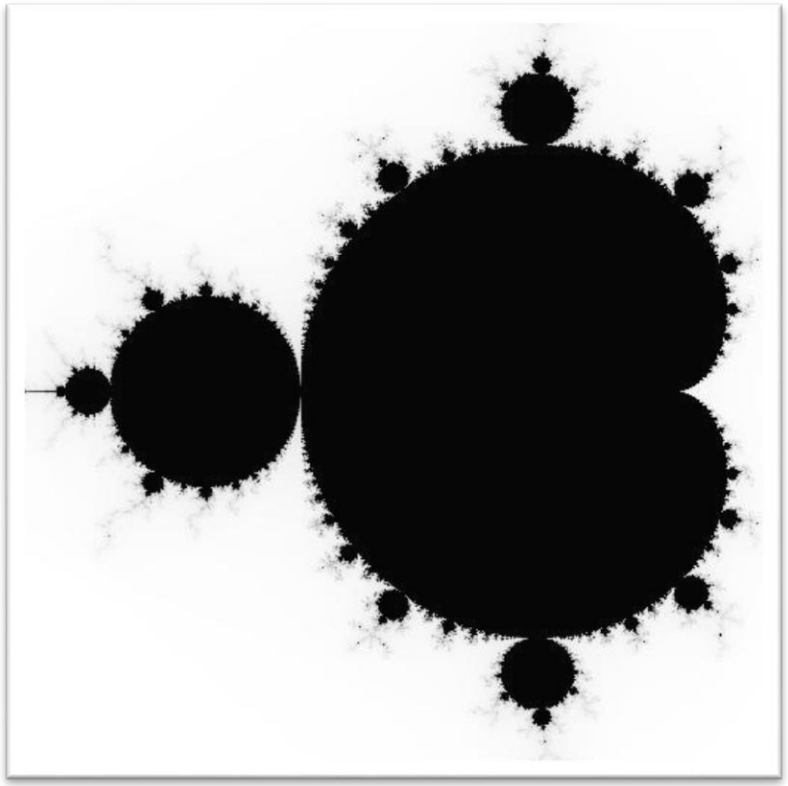
```
public static void main(String[] args) {
   double xc   = Double.parseDouble(args[0]);
   double yc   = Double.parseDouble(args[1]);
   double size = Double.parseDouble(args[2]);
   int N = 512;
   Picture pic = new Picture(N, N);
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         double x0 = xc - size/2 + size*i/N;
         double y0 = yc - size/2 + size*j/N;
         Complex z0 = new Complex(x0, y0);
         Color color = mand(z0);
         pic.set(i, N-1-j, color);
      }
   }
   pic.show();
}
```

در نظر گرفتن گوشه بالا سمت چپ به عنوان مبدأ مختصات

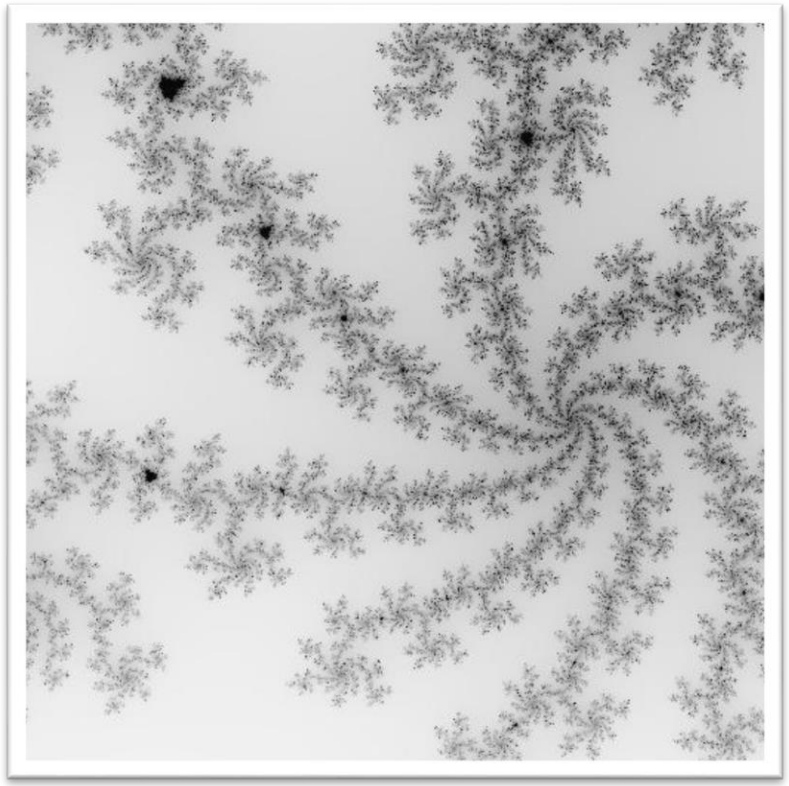


```
% java Mandelbrot .1045 -.637 .01
```

مجموعه مندلبرو



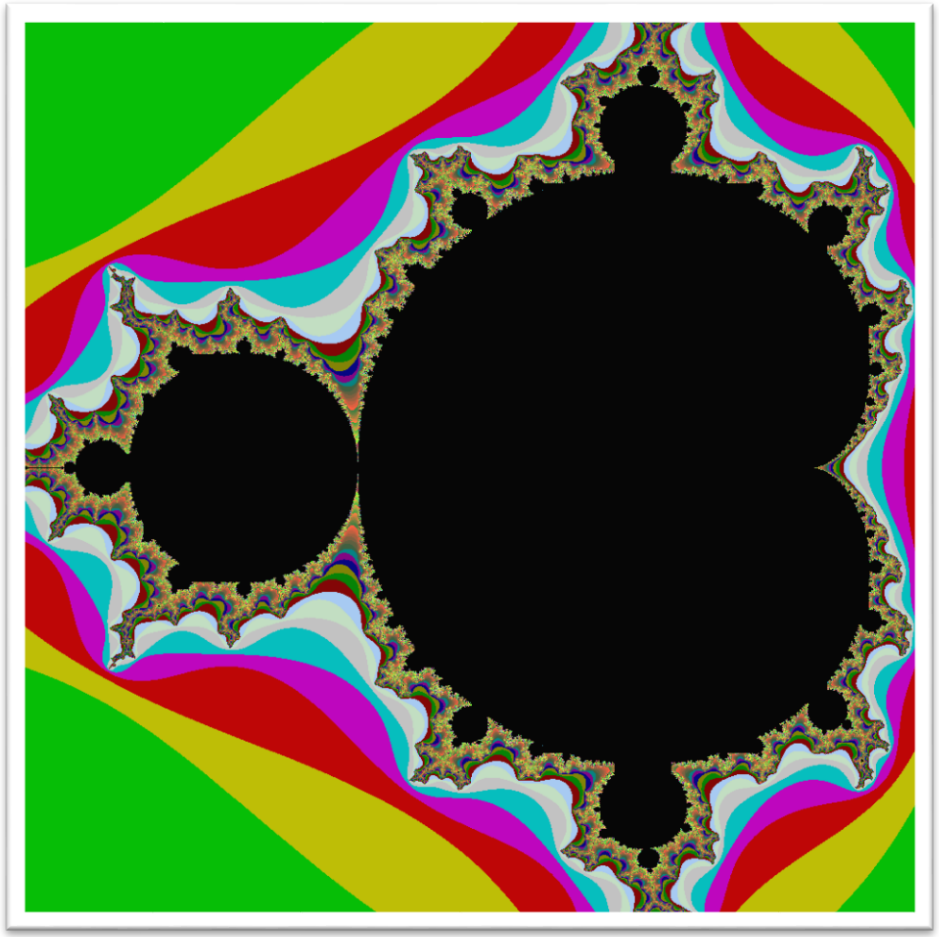
% java Mandelbrot -.5 0 2



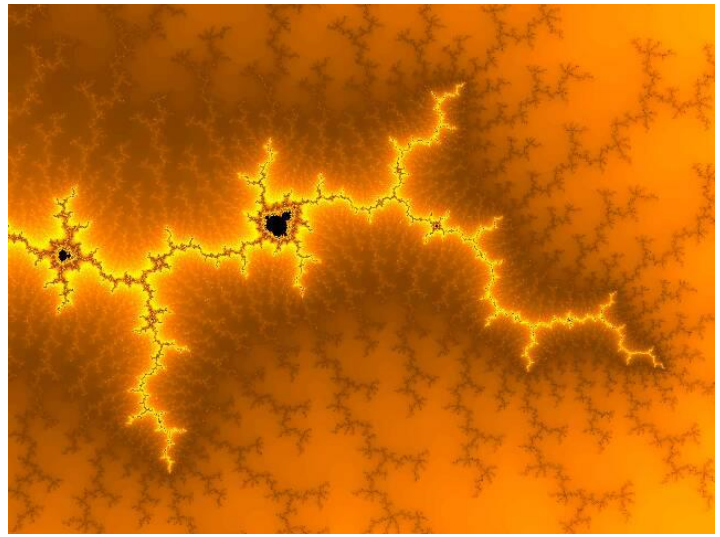
% java Mandelbrot .1045 -.637 .01

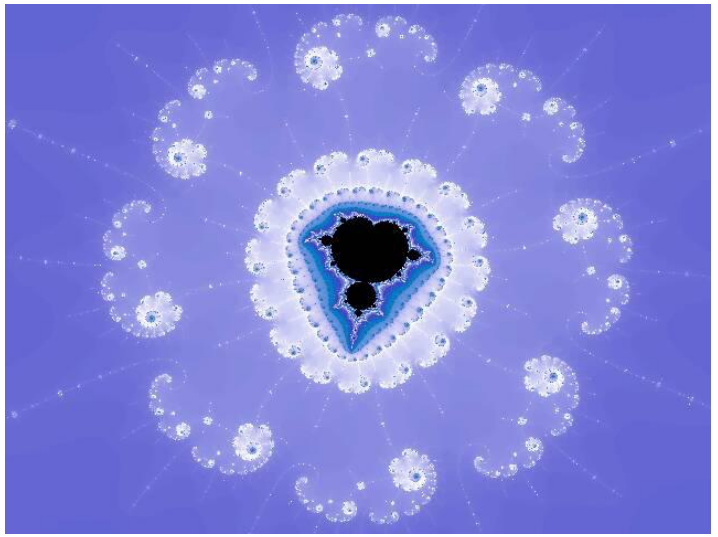
مجموعه مندلبرو


```
% java ColorMandelbrot -.5 0 2 < mandel.txt
```

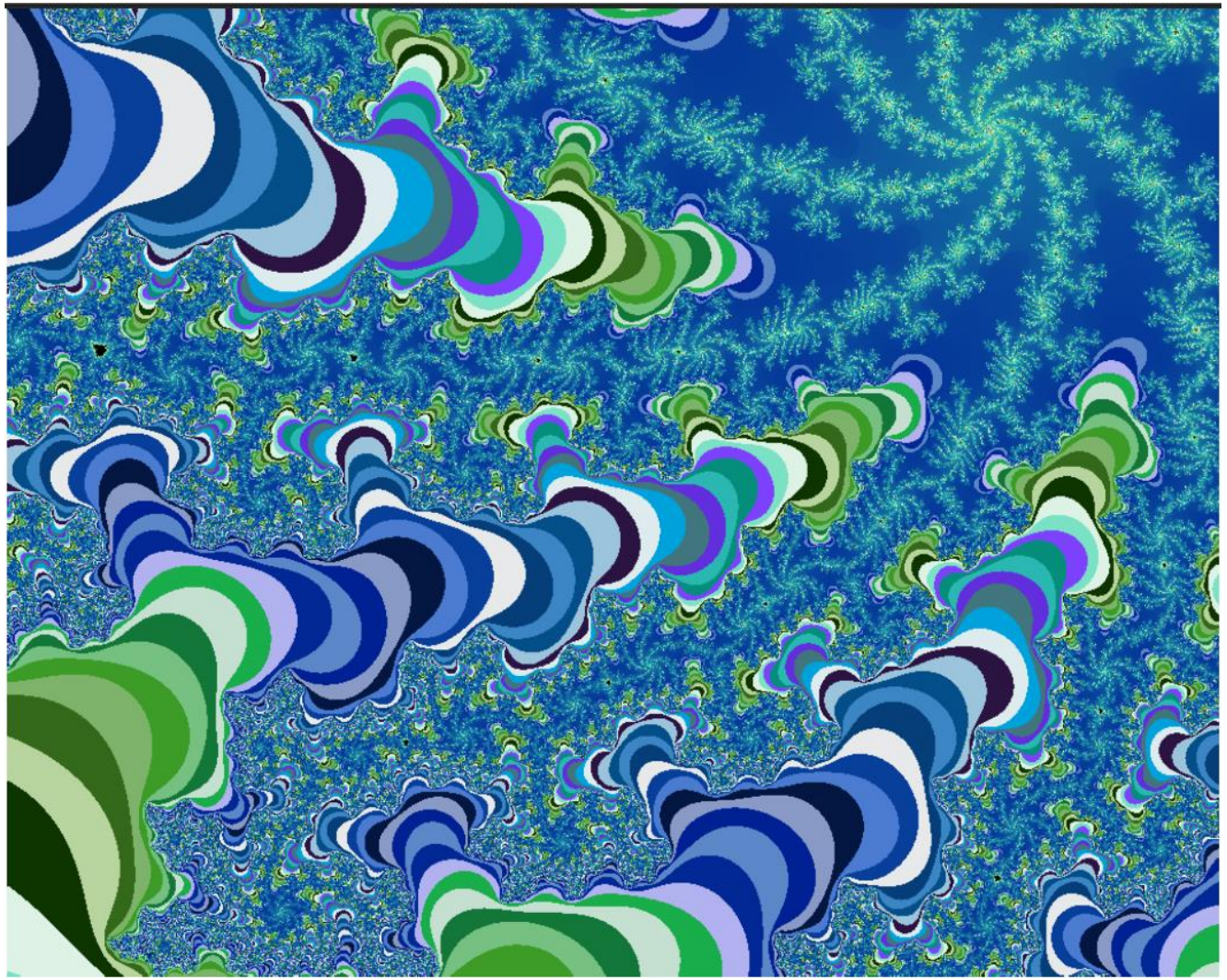


مجموعه مندلبرو





مجموعه مندلبرو



انواع داده‌ای: کاربردها

- نوع داده‌ای. یک مجموعه از مقادیر به همراه عملیات قابل انجام بر روی آن مجموعه.

- شبیه‌سازی دنیای فیزیکی.
  - اشیای جاوا مدل کننده اشیای دنیای واقعی هستند.
  - ایجاد مدل‌هایی که بیانگر واقعیت هستند، همیشه ساده نیست.
  - مثال: ذره باردار، مولکول، دانشجوی کلاس جاوا، ...

- گسترش دادن زبان جاوا.
  - در جاوا همه اشیای مورد نیاز در کاربردهای گوناگون پیش‌بینی نشده است.
  - انواع داده‌ای به ما امکان می‌دهند انواع مورد نیازمان را خودمان ایجاد کنیم.
  - مثال: عدد مختلط، بردار، چندجمله‌ای، ماتریس، ...