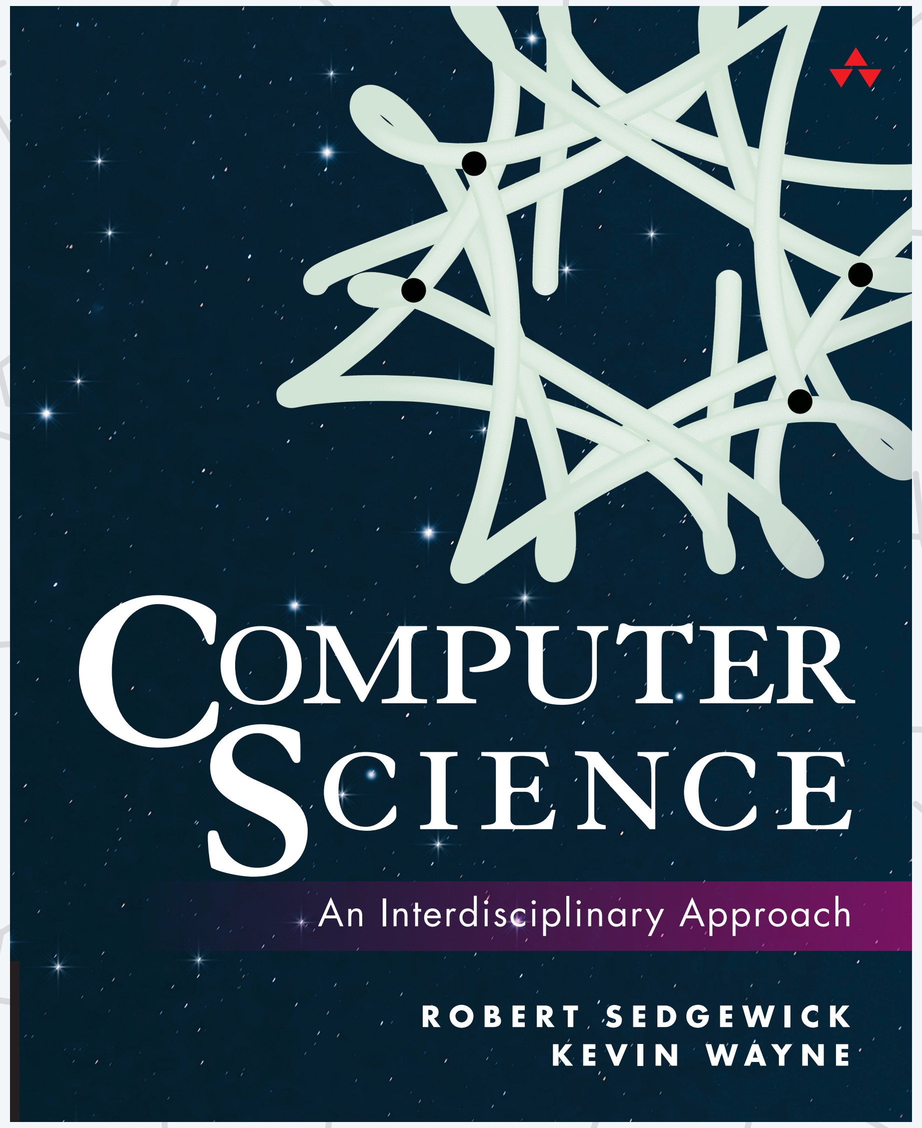
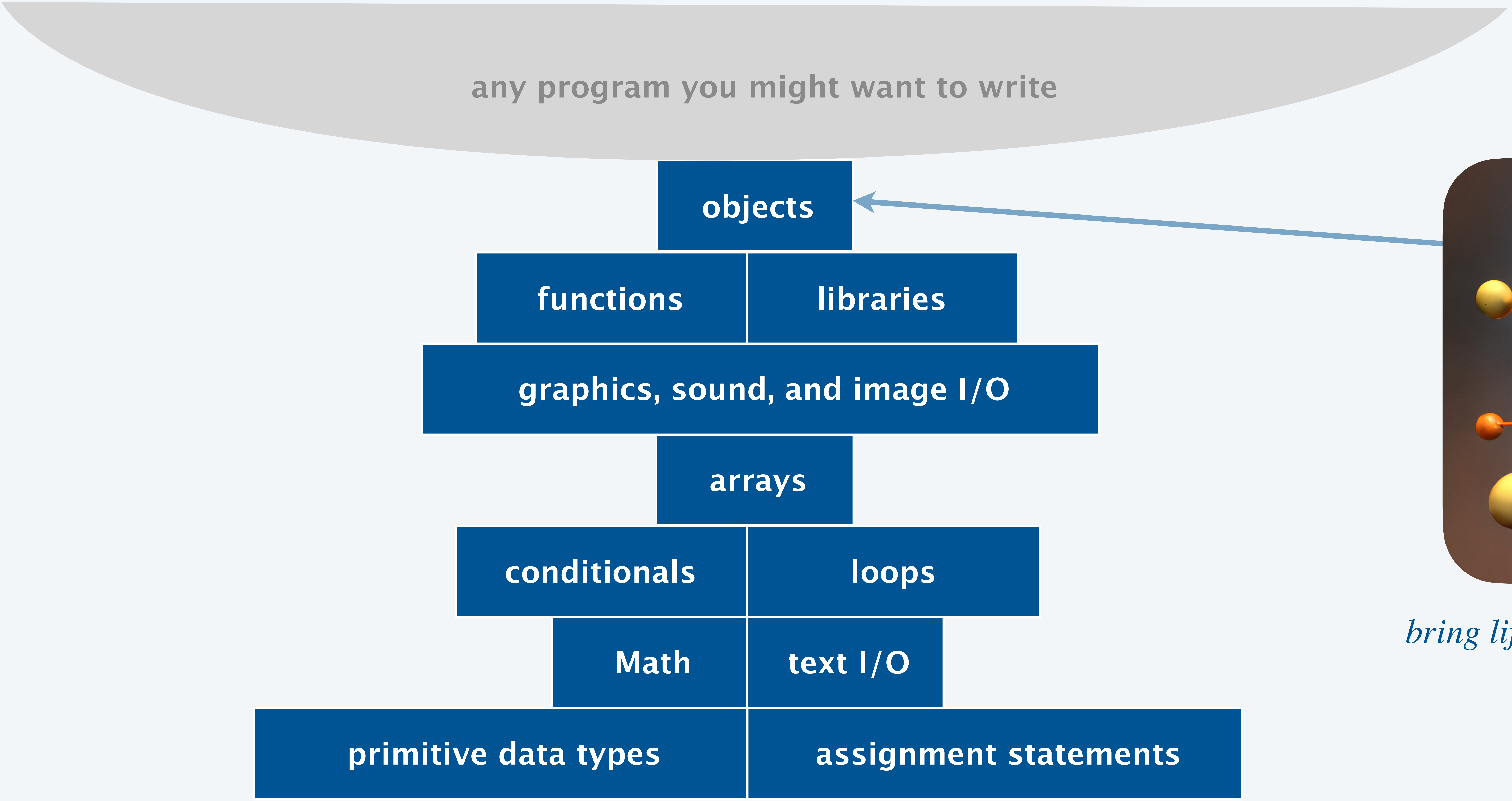


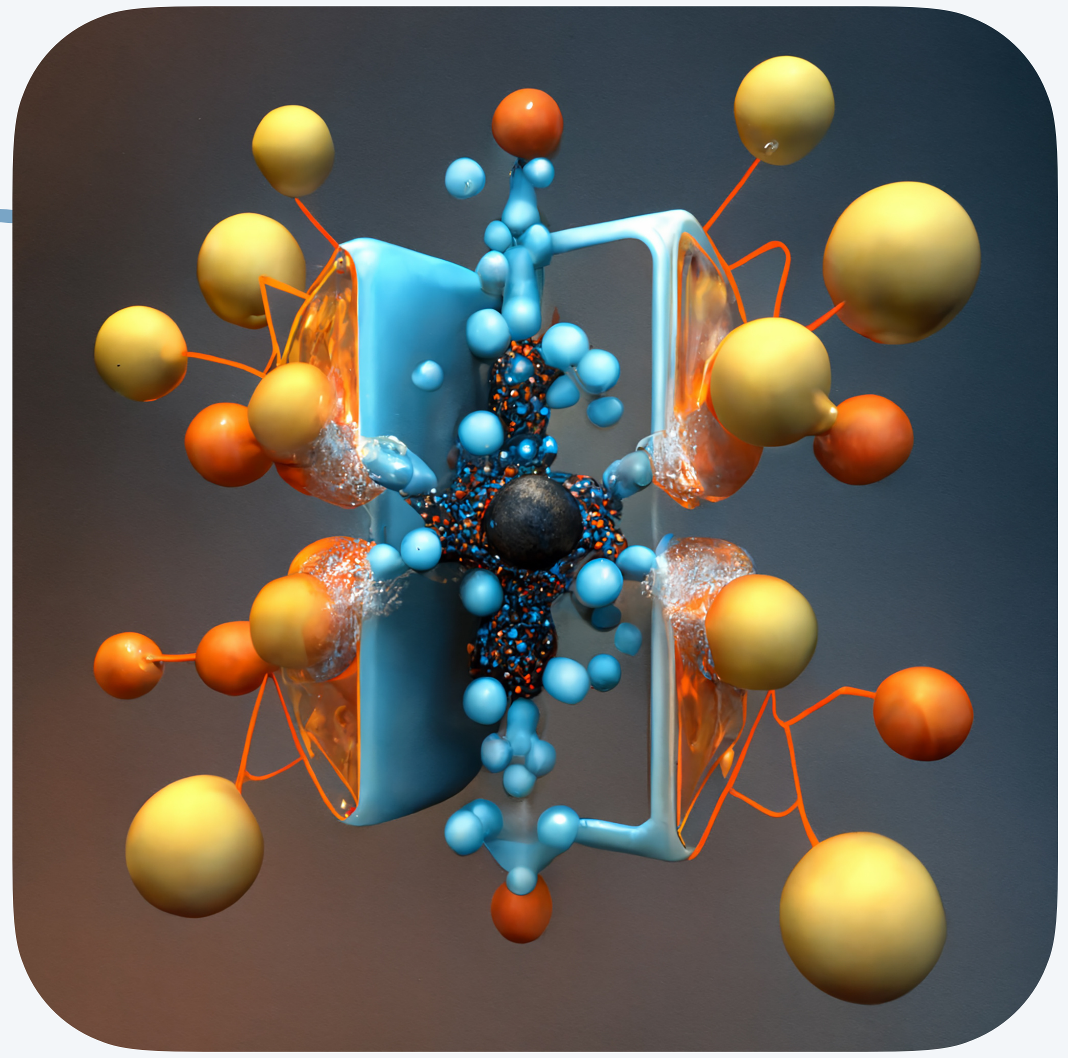
https://introcs.cs.princeton.edu

3.2 Creating Data Types

- point data type
- circle data type
- clock data type
- complex number data type

Basic building blocks for programming

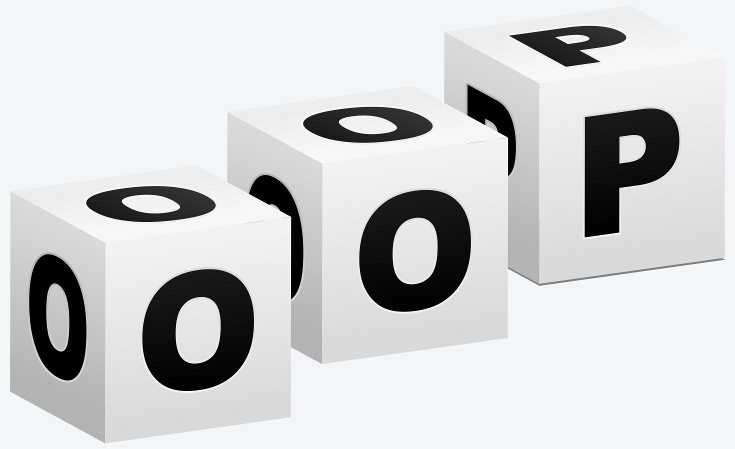




bring life to your own abstractions

Object-oriented programming (OOP)

A data type is a set of values and a set of operations on those values.



We want to write programs that process other types of data.

- Strings, colors, pictures, …
- Points, circles, complex numbers, vectors, matrices, …
- GUIs, database connections, neural networks, plots, …

Last lecture. Use pre-existing data types.

This lecture. Create your own data types.

data type	**set of values**	**example values**	**operations**
`String`	*sequences of characters*	`"Hello, World"` `“COS 126 is fun"`	*concatenate, length, substring, …*
`Complex`	*complex numbers*	$3 + 5i$ $-5 + 4i$	*add, multiply, magnitude, …*

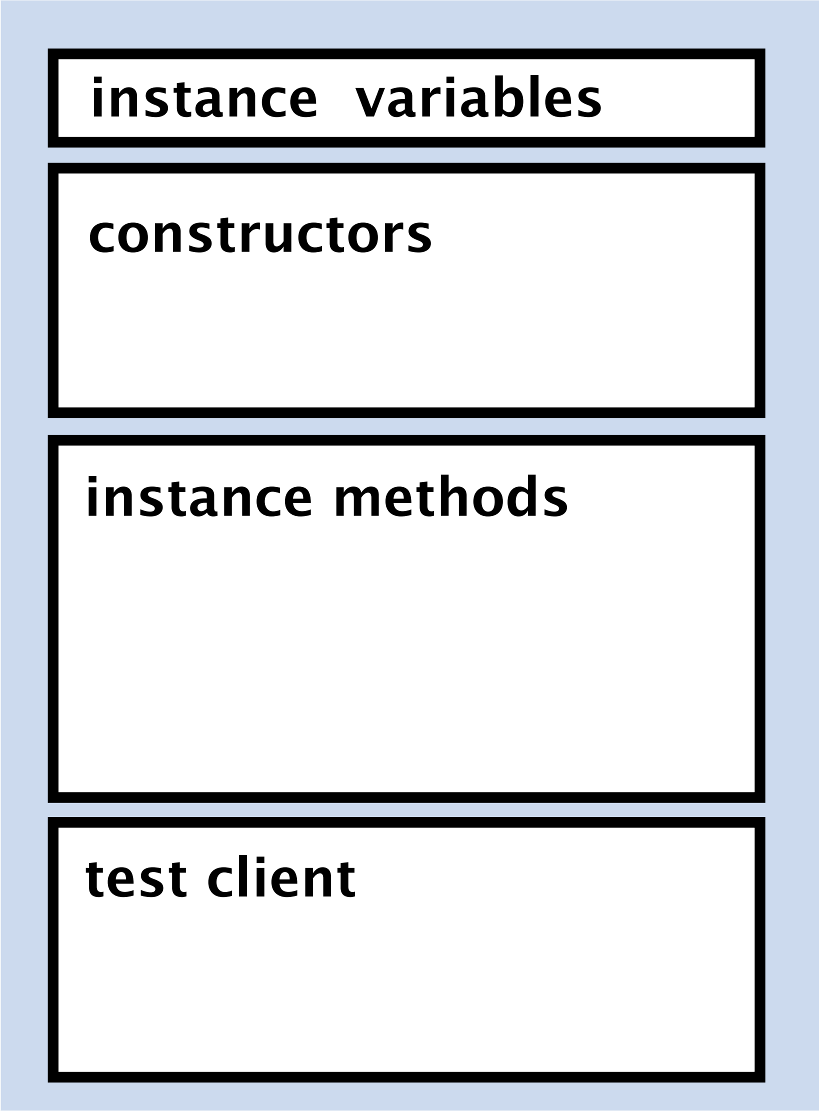
Implementing a data type

A data type is a set of values and a set of operations on those values.

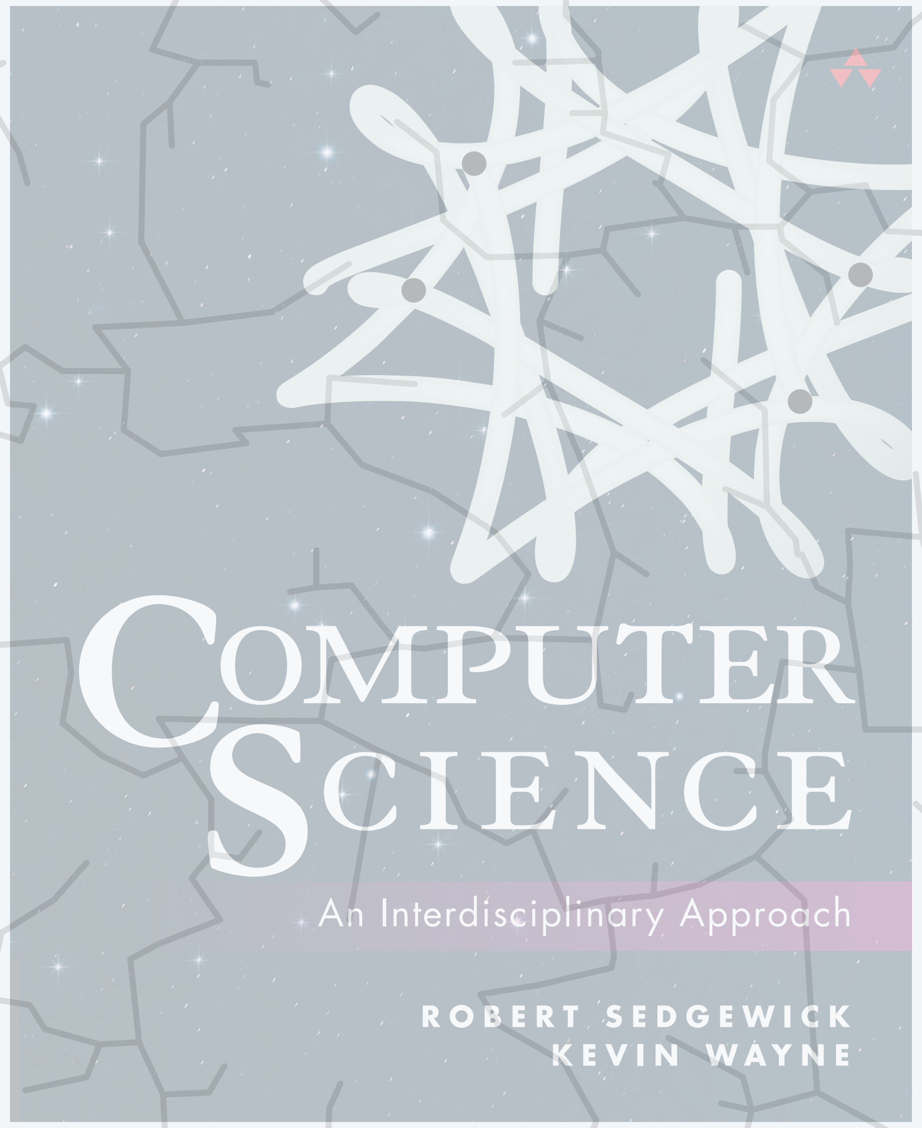
Implementing a data type. Provide code that:

- Defines the set of values (instance variables).
- Implements operations on those values (instance methods).
- Creates and initialize new objects (constructors).

In Java, you implement a data type in a class.



Java class



https://introcs.cs.princeton.edu

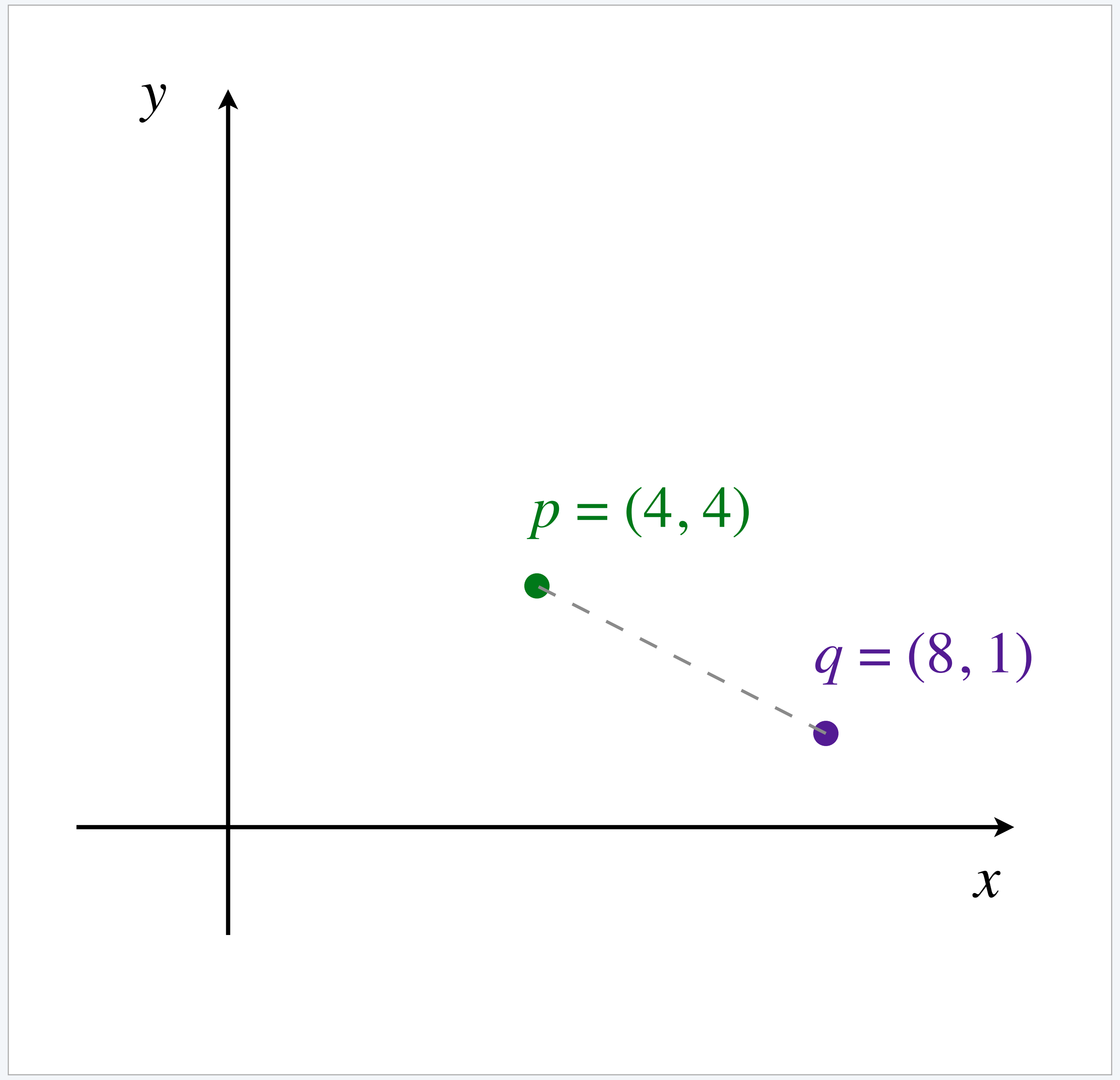
3.2 Creating Data Types

- point data type
- circle data type
- clock data type
- complex number data type

A data type for points

A **2d point** is a location in the plane.

The *Point* data type allows us to write programs that manipulate points.



	point	location (x, y)
values	p	$(4, 4)$
	q	$(8, 1)$

API

`public class Point`		description
	`Point(double x0, double y0)`	*create point* (x_0, y_0)
`double`	`distanceTo(Point other)`	*Euclidean distance between two points*
`String`	`toString()`	*string representation of this point*

Point implementation: test client

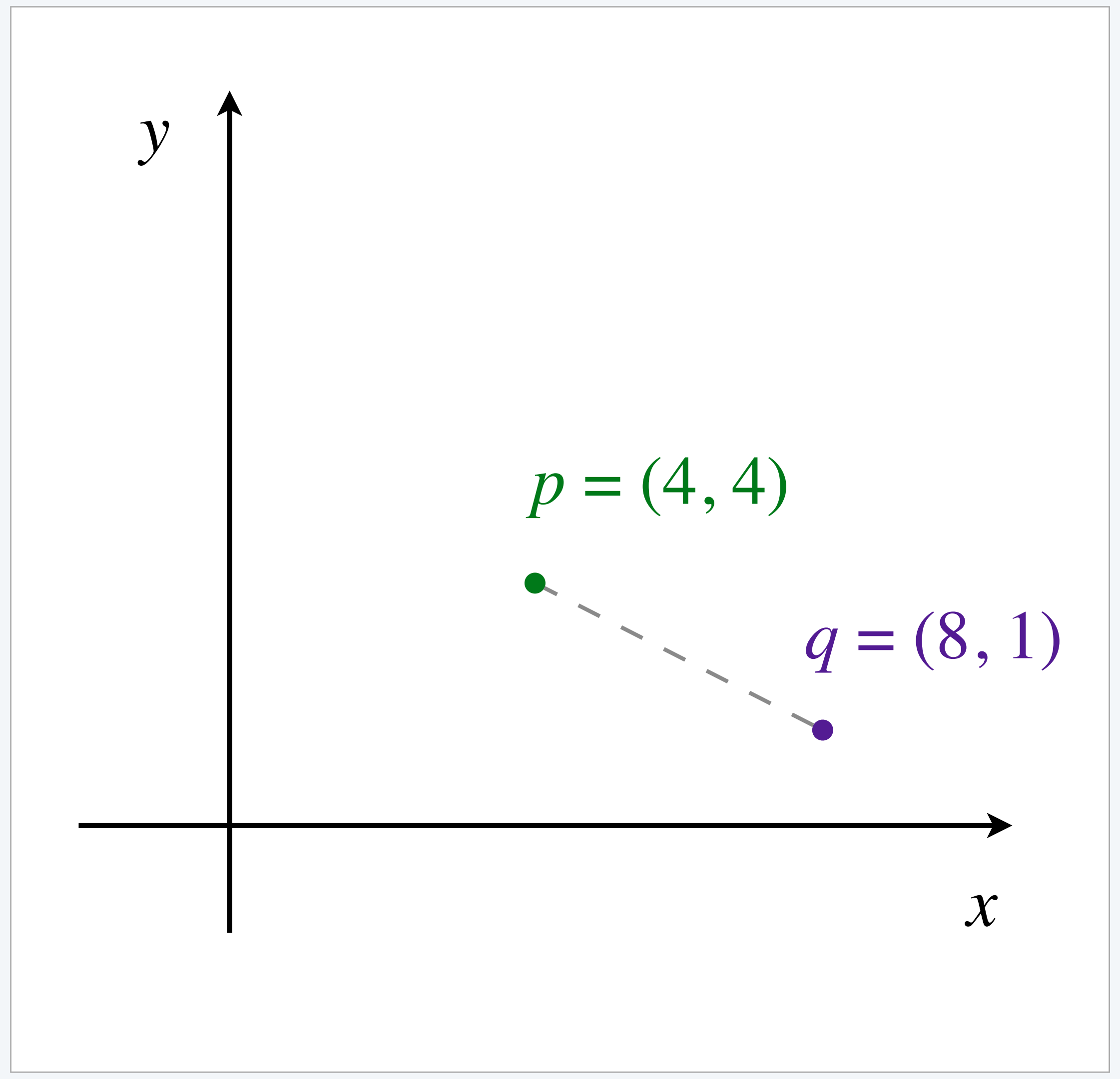
Best practice. Begin by implementing a simple test client.

```
public static void main(String[] args) {
   Point p = new Point(4.0, 4.0);
   Point q = new Point(8.0, 1.0);
   StdOut.println("p  = " + p);
   StdOut.println("q  = " + q);
   StdOut.println("dist(p, q) = " + p.distanceTo(q));
}
```

automatically calls `p.toString()`

```
~/cos126/oop1> javac-introcs Point.java
~/cos126/oop1> java-introcs Point
p  = (4.0, 4.0)
q  = (8.0, 1.0)
dist(p, q) = 5.0
```

desired output



$$r = \sqrt{(x_2 - x_1)^2 \ + \ (y_2 - y_1)^2}$$
$$= \sqrt{(4)^2 + (-3)^2}$$
$$= 5$$

instance variables

constructors

instance methods

test client

Point implementation: instance variables

Instance variables. Define data type values.

Internal representation. Two real numbers (position).

each point has it own position (so needs its own variables)

class name matches name of file (`Point.java`)

declared inside the class (but outside any method)

```
public class Point {
    private final double x;   // x-coordinate
    private final double y;   // y-coordinate
    ...
}
```

instance variables

constructors

instance methods

test client

Private access modifier. Helps enforce encapsulation.

Final access modifier. Helps enforce immutability.

stay tuned (next lecture)

Point implementation: constructor

Constructor. Create and initialize new objects.

- Name is same as class.
- Similar to void method (arguments and body).
- But can refer to instance variables (and no *static* or *void* keywords).

```
public class Point {
   private final double x;   // x-coordinate
   private final double y;   // y-coordinate

   public Point(double x0, double y0) {
      x = x0;
      y = y0;
   }

   ...
}
```

instance variables of object being constructed

constructor name matches name of class

instance variables

constructors

instance methods

test client

Point implementation: instance methods

Instance methods. Define data-type operations.

- Similar to static methods (arguments, return type, and body).
- But can refer to instance variables (and no *static* keyword).

```
public class Point {
   ...
   // returns the Euclidean distance between the two points
   public double distanceTo(Point other) {

      double dx = other.x - x;
      double dy = other.y - y;
      return Math.sqrt(dx*dx + dy*dy);
   }

   // returns a string representation of this point
   public String toString() {
      return "(" + x + ", " + y + ")";
   }

   ...
}
```

instance variable of argument object

instance variable of invoking object

instance variables

constructors

instance methods

test client

Anatomy of a Java class

text file named `Point.java`

instance variables

constructor

instance methods

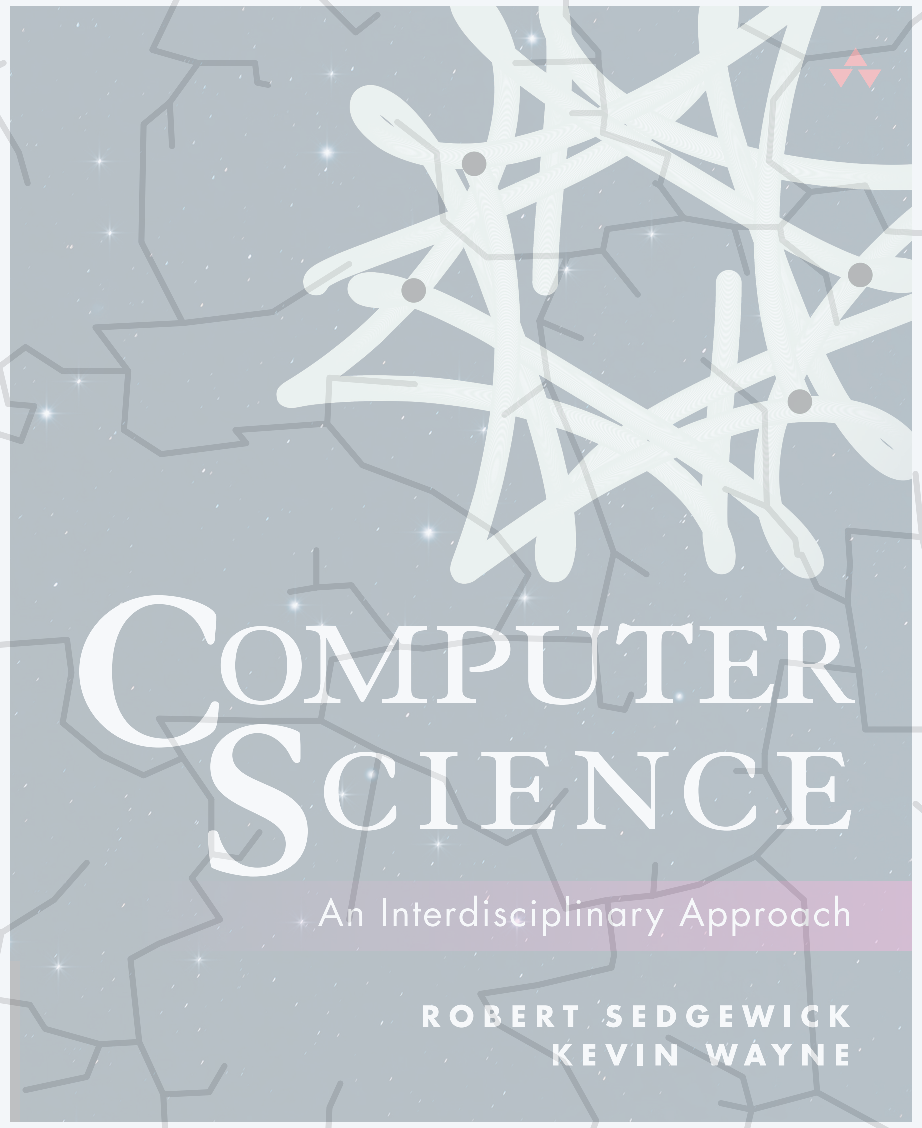
test client

```
public class Point {
    private final double x;    // x-coordinate
    private final double y;    // y-coordinate

    // creates and initializes a point with given (x0, y0)
    public Point(double x0, double y0) {
        x = x0;
        y = y0;
    }

    // return the Euclidean distance between the two points
    public double distanceTo(Point other) {
        double dx = other.x - x;
        double dy = other.y - y;
        return Math.sqrt(dx*dx + dy*dy);
    }

    // return string representation of this point
    public String toString() {
        return "(" + x + ", " + y + ")";
    }

    public static void main(String[] args) {
        Point p = new Point(4.0, 4.0);
        Point q = new Point(8.0, 1.0);
        StdOut.println("p  = " + p);
        StdOut.println("q  = " + q);
        StdOut.println("dist(p, q) = " + p.distanceTo(q));
    }
}
```

Suppose that you make the follow modification to the constructor. What is the effect?

- **A.** Still works.
- **B.** The x– and y–coordinates are initialized to 0.
- **C.** Run–time error.
- **D.** Compile–time error.

```
public class Point {
   private double x;   // x-coordinate
   private double y;   // y-coordinate

   public Point(double x0, double y0) {
      double x = x0;
      double y = y0;
   }

   ...
}
```



https://introcs.cs.princeton.edu

3.2 Creating Data Types

- point data type
- circle data type
- clock data type
- complex number data type

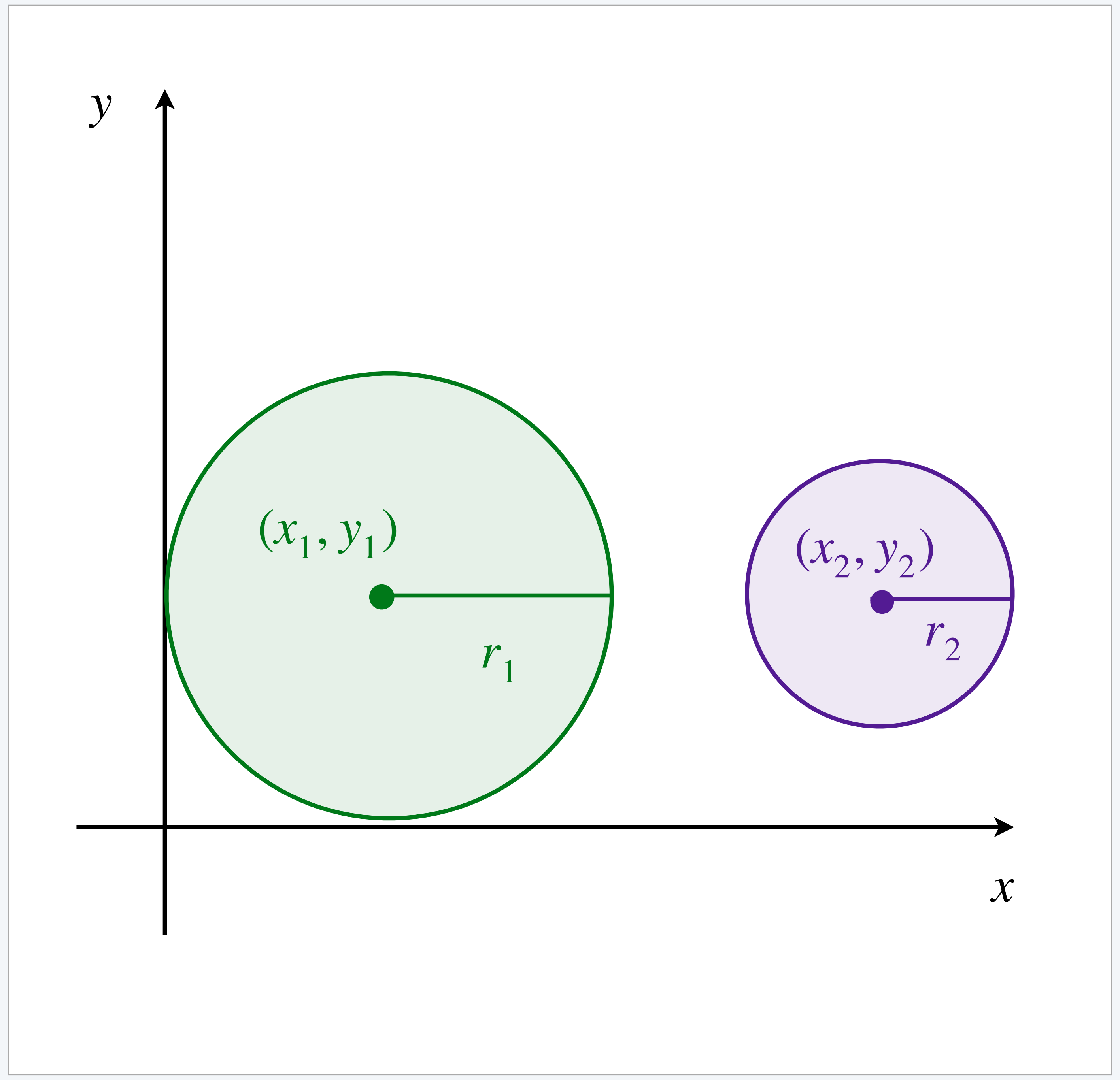
A data type for circles

A circle is the set of all points that are at a given distance from a point.

The *`Circle`* data type us to write programs that manipulate circles.

values

circle	location (x, y)	radius (r)
c_1	$(2, 2)$	2
c_2	$(6, 2)$	1



API

`public class Circle`	**description**
`Circle(Point c, double r)`	*create circle with center* `c` *and radius* `r`
`double area()`	*area of this circle*
`boolean contains(Point p)`	*is point* `p` *inside the circle?*
`String toString()`	*string representation of this circle*

Circle implementation: test client

Best practice. Begin by implementing a simple test client.

```
public static void main(String[] args) {
   Point p = new Point(5.0, 5.0);
   Circle c1 = new Circle(2.0, 2.0, 2.0);
   Circle c2 = new Circle(6.0, 2.0, 1.0);
   StdOut.println("p  = " + p);
   StdOut.println("c1 = " + c1);
   StdOut.println("c2 = " + c2);
   StdOut.println("area(c2)           = " + c2.area());
   StdOut.println("contains(c1, p)    = " + c1.contains(p));
}
```

automatically invokes `c.toString()`

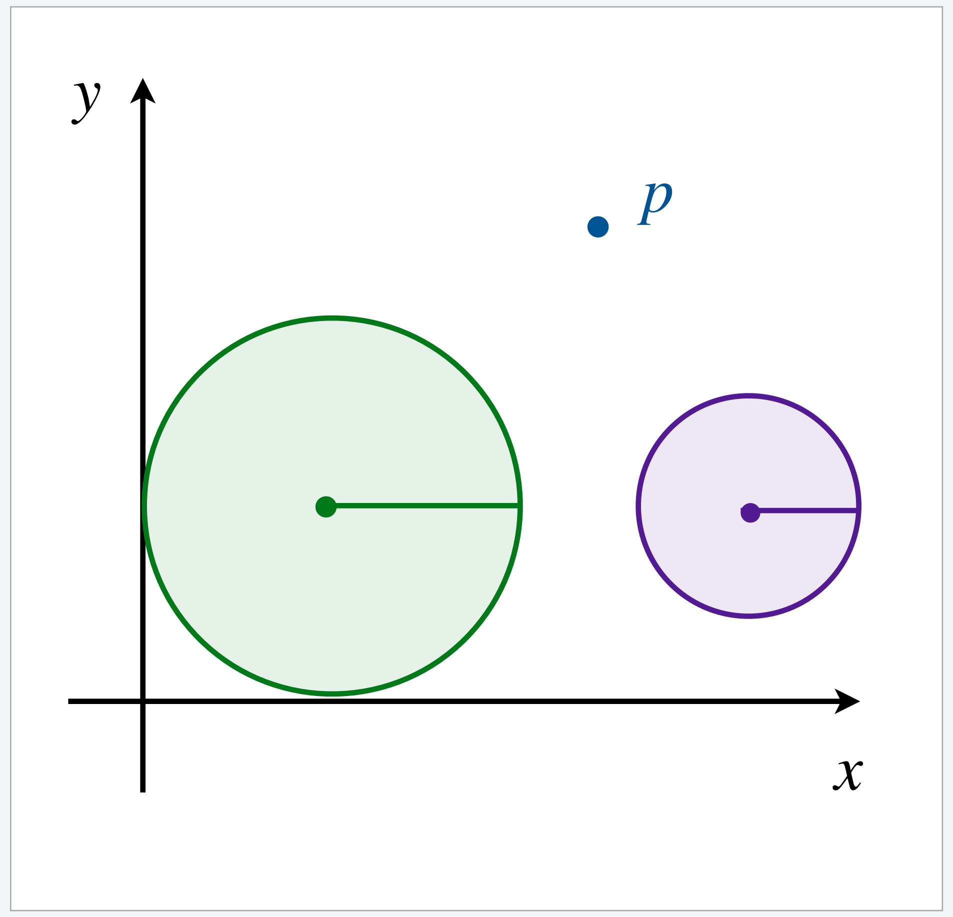
```
~/cos126/oop2> java-introcs Circle
p  = (5.0, 5.0)
c1 = (2.0, 2.0), 2.0
c2 = (6.0, 2.0), 1.0
area(c2)           = 3.141592653589793
contains(c1, p)    = false
```

instance variables

constructors

instance methods

test client



two circles and a point

Circle implementation: instance variables

Instance variables. Define data type values.

Internal representation. A point (center) and a real number (radius).

instance variable is of type Point

```
public class Circle {
   private final Point center;    // center of circle
   private final double radius;   // radius of circle
   ...
}
```

The type of an instance variable can be any

- Primitive type. ← int, double, boolean, ...
- Built-in reference type. ← String, Color, int[], ...
- User-defined reference type. ← Point, Circle, Picture, ...

Circle implementation: constructor

Constructor. Create and initialize new objects.

instance variables
constructors
instance methods
test client

```
public class Circle {
   private final Point center;   // center of circle
   private final double radius;  // radius of circle

   public Circle(double x, double y, double r) {
      center = new Point(x, y);
      radius = r;
   }

   ...
}
```

instance variables of object being constructed

Circle implementation: instance methods

Instance methods. Define data-type operations.

```
public class Circle {
   ...
   // area of this circle
   public double area() {
      return Math.PI * radius * radius;
   }

   // is the point p contained inside this circle?
   public boolean contains(Point p) {
      return p.distanceTo(center) <= radius;
   }

   // string representation of this circle
   public String toString() {
      return center + ", " + radius;
   }

   ...
}
```

takes a Point *object as argument*

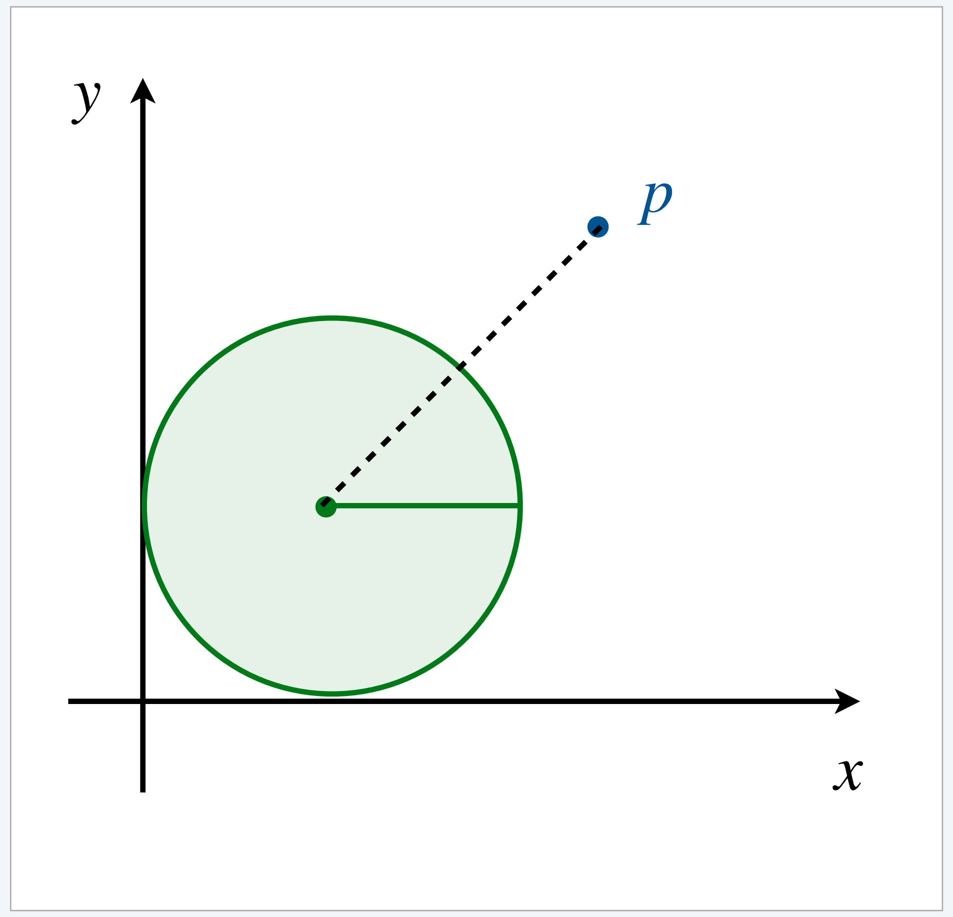
calls a Point *instance method*

instance variables

constructors

instance methods

test client



circle contains point if distance from p to center ≤ radius

Circle implementation

text file named `Circle.java`

instance variables

constructor

instance methods

```
public class Circle {

    private final Point center;   // center of circle
    private final double radius;  // radius of circle

    public Circle(double x, double y, double r) {
        center = new Point(x, y);
        radius = r;
    }

    // area of this circle
    public double area() {
        return Math.PI * radius * radius;
    }

    // is the point p contained inside this circle?
    public boolean contains(Point p) {
        return p.distanceTo(center) <= radius;
    }

    // string representation of this circle
    public String toString() {
        return center + ", " + radius;
    }

}
```

How to implement a method that checks whether two circles intersect?

A.

```
public boolean intersects(Circle circle) {
   return center.distanceTo(circle.center) <= radius + circle.radius;
}
```

B.

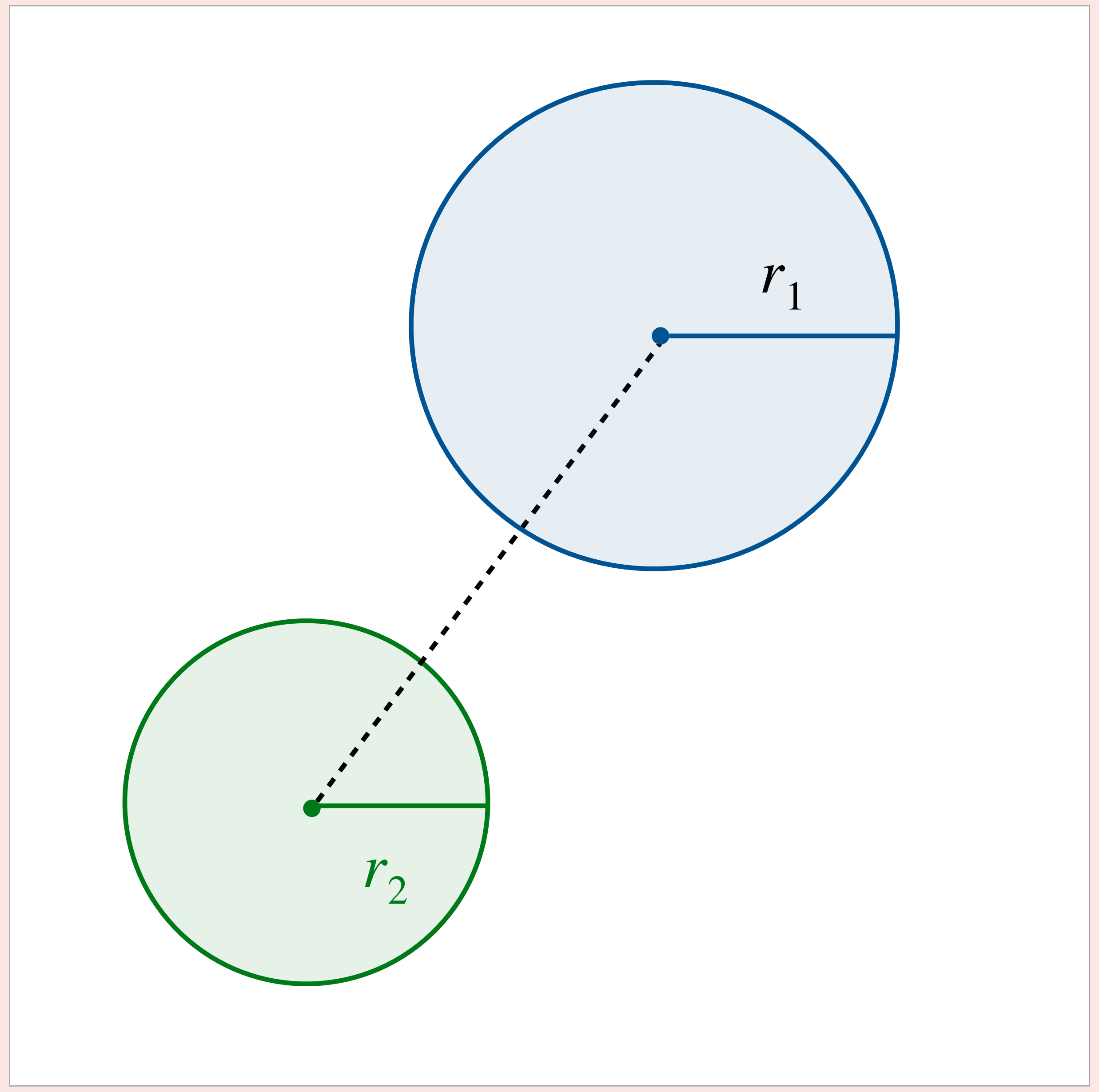
```
public boolean intersects(Circle circle) {
   return circle.distanceTo(center) <= radius + circle.radius;
}
```

no such instance method in `Circle` class

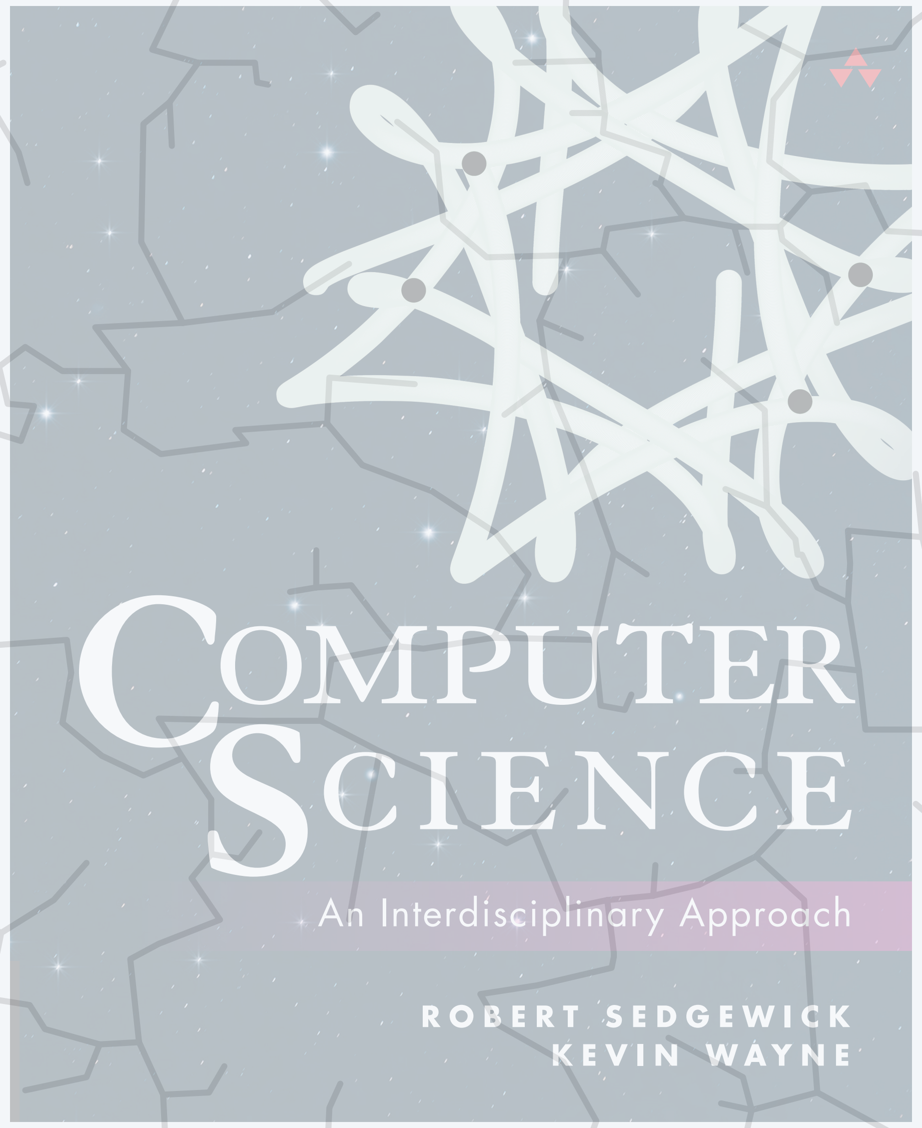
can access instance variables of any object in same class

C. Both A and B.

D. Neither A nor B.



two circles intersect if the distance between their centers ≤ sum of their radii



https://introcs.cs.princeton.edu

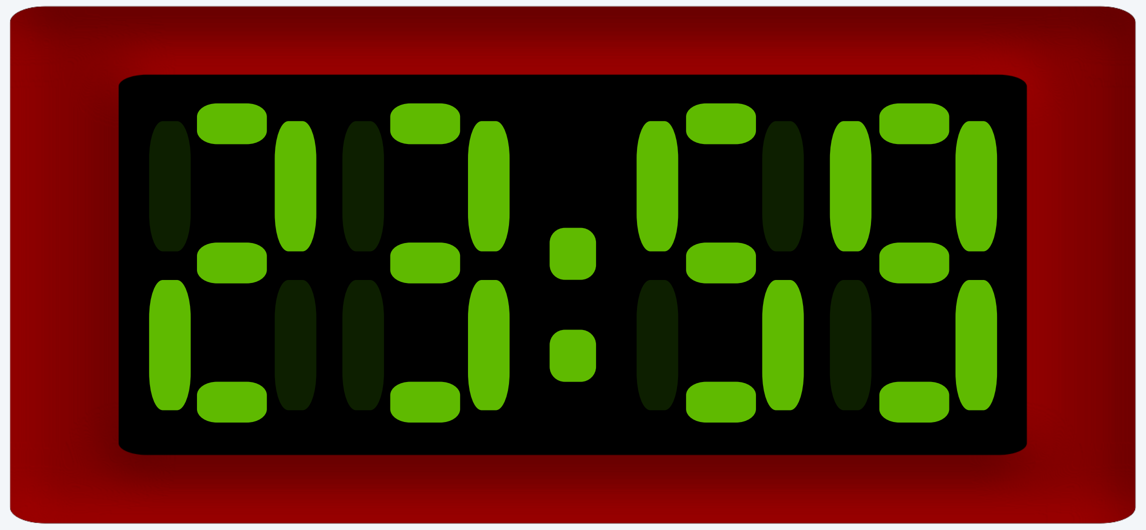
3.2 Creating Data Types

- point data type
- circle data type
- clock data type
- complex number data type

24-hour clock

A 24-hour clock displays the time in *hh:mm* format.

24-hour clock	12-hour clock	description
00:00	12:00am	*midnight*
12:00	12:00pm	*noon*
23:59	11:59pm	*one minute before midnight*
01:00	1:00am	*one hour after midnight*
13:26	1:26pm	*4 minutes before class starts*
24:01	–	*invalid time*



24-hour clock API

A 24-hour clock displays the time in *hh:mm* format.

values

time	hours	minutes
13:26	13	26
23:59	23	59

API

public class Clock	description
`Clock(int h, int m)`	*create clock with* `h` *hours and* `m` *minutes*
`void tic()`	*advance the time by one minute*
`boolean isEarlierThan(Clock other)`	*is the time of this clock earlier than* `other`
`String toString()`	*string representation of this clock*
`void speak()`	*say the time*
`void draw()`	*draw the clock*

mutable (data-type value can change)

Clock implementation: test client

Best practice. Begin by implementing a simple test client.

```
public static void main(String[] args) {
   Clock now = new Clock(13, 30);
   Clock end = new Clock(14, 50);
   while (now.isEarlierThan(end)) {
      StdOut.println(now);
      now.tic();
   }
}
```

```
~/cos126/oop2> java-introcs Clock
13:30
13:31
13:32
...
14:48
14:49
```

instance variables
constructors
instance methods
test client

Clock implementation: instance variables

Instance variables. Define data type values.

Internal representation. Two integers (hours and minutes).

each clock has it own time (so needs its own variables)

```
public class Clock {

   private int hours;     // hours   (0 to 23)
   private int minutes;   // minutes (0 to 59)

   ...
}
```

one variable per object

Clock implementation: constructor

Constructors. Create and initialize new objects.

```
public class Clock {

   private int hours;    // hours   (0 to 23)
   private int minutes;  // minutes (0 to 59)

   public Clock(int h, int m) {
      hours = h;
      minutes = m;
   }

   ...
}
```

instance variables

constructors

instance methods

test client

Clock implementation: instance methods

Instance methods. Define data-type operations.

```java
public class Clock {
   private static final int MINUTES_PER_HOUR = 60;
   private static final int HOURS_PER_DAY = 24;
   ...

   // increment the time by 1 minute
   public void tic() {
      minutes++;
      if (minutes == MINUTES_PER_HOUR) {
         minutes = 0;
         hours++;
      }
      if (hours == HOURS_PER_DAY) {
         hours = 0;
      }
   }

   ...
}
```

class constants
(one variable per class)

instance variables

constructors

instance methods

test client

Clock implementation: instance methods

Instance methods. Define data-type operations.

```
public class Clock {
   ...

// is this clock earlier than the other one?
public boolean isEarlierThan(Clock other) {
   if (hours < other.hours) return true;
   if (hours > other.hours) return false;
   return minutes < other.minutes;
}

// string representation, using format HH:MM
public String toString() {
   return String.format("%02d:%02d", hours, minutes);
}

   ...
}
```

`format()` *works like* `printf()`, *but returns formatted string (instead of printing it)*

instance variables

constructors

instance methods

test client

Clock implementation

text file named `Clock.java`

class constants

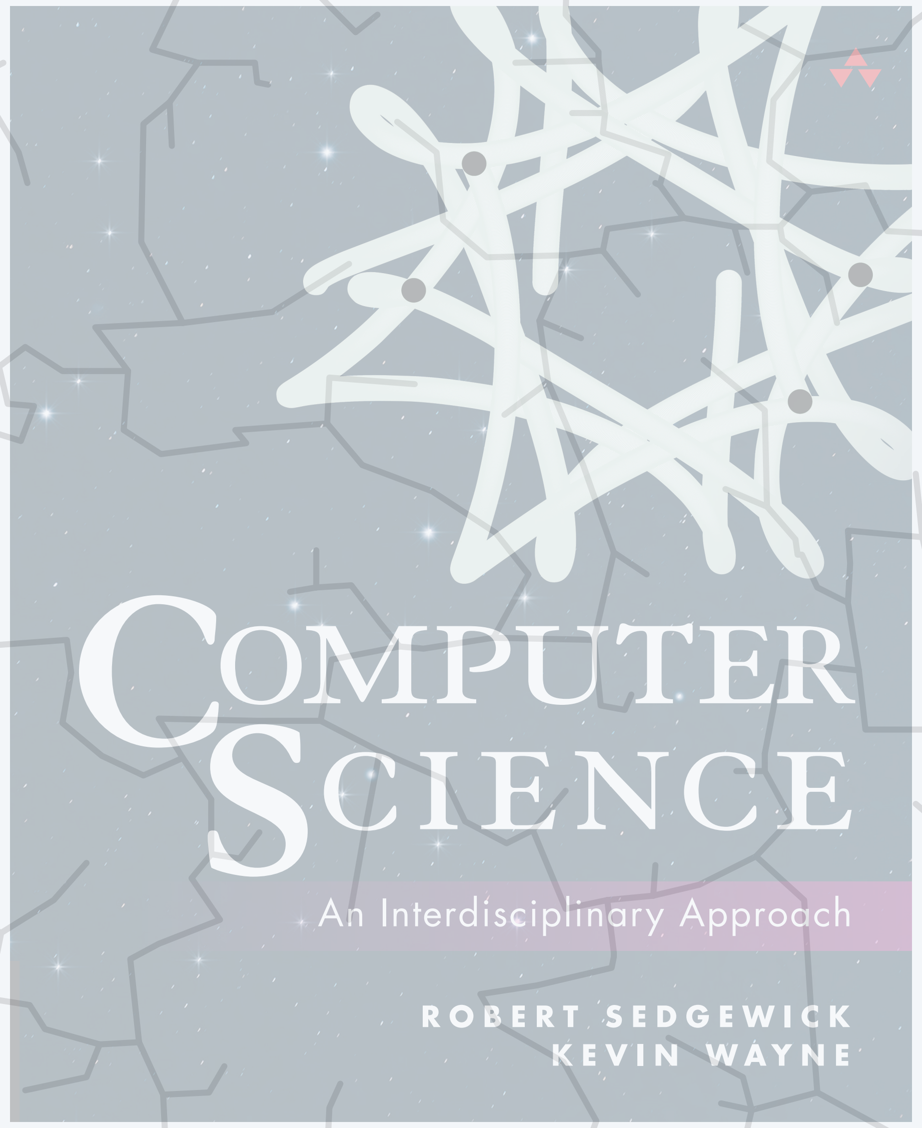
instance variables

constructor

instance methods

```
public class Clock {
   private static final int MINUTES_PER_HOUR = 60;
   private static final int HOURS_PER_DAY = 24;

   private int hours;     // hours   (between 0 and 23)
   private int minutes;   // minutes (between 0 and 59)

   public Clock(int h, int m) {
      hours = h;
      minutes = m;
   }

   // increment the time by 1 minute
   public void tic() {
      minutes++;
      if (minutes == MINUTES_PER_HOUR) {
         minutes = 0;
         hours++;
      }
      if (hours == HOURS_PER_DAY) {
         hours = 0;
      }
   }

   // string representation, using format HH:MM
   public String toString() {
      return String.format("%02d:%02d", hours, minutes);
   }
```

```
   ...

   // is this clock earlier than the other one?
   public boolean isEarlierThan(Clock other) {
      if (hours < other.hours) return true;
      if (hours > other.hours) return false;
      return minutes < other.minutes;
   }

   public static void main(String[] args) {
      Clock clock1 = new Clock(13, 26);
      Clock clock2 = new Clock(13, 26);
      for (int time = 0; time < 1440; time++) {
         clock1.tic();
         clock2.tic();
         StdOut.println(clock1);
      }
   }
}
```

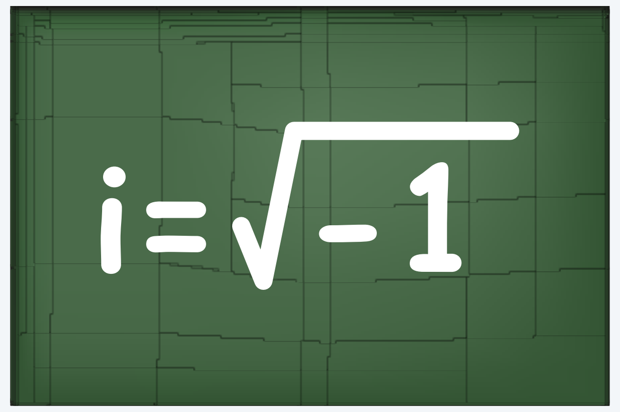


https://introcs.cs.princeton.edu

3.2 Creating Data Types

- *point data type*
- *circle data type*
- *clock data type*
- *complex number data type*

Crash course in complex numbers



A complex number is a number of the form $a + bi$, where a and b are real and $i = \sqrt{-1}$.

- Quintessential mathematical abstraction.
- Applications in STEM: signal processing, electrical circuits, quantum mechanics, …

Operations on complex numbers.

- Addition: $(a + bi) + (c + di) \; = \; (a + c) + (b + d)i$.
- Multiplication: $(a + bi) \times (c + di) \; = \; (ac - bd) \; + \; (bc + ad)i$.
- Magnitude: $\left| a + bi \right| \; = \; \sqrt{a^2 + b^2}$
- …

operation	result
$(3 + 4i) \; + \; (-2 + 3i)$	$1 + 7i$
$(3 + 4i) \; \times \; (-2 + 3i)$	$-18 + i$
$\left| 3 + 4i \right|$	5

Data type for complex numbers

A complex number is a number of the form $a + bi$, where a and b are real and $i = \sqrt{-1}$.

The *Complex* data type allows us to write programs that manipulate complex numbers.

values

complex number
$3 + 4i$
$-2 + 2i$
$126i$

API

`public class Complex`	description
`Complex(double real, double imag)`	*create a new complex number*
`Complex plus(Complex b)`	*sum of this number and* `b`
`Complex times(Complex b)`	*product of this number and* `b`
`double abs()`	*magnitude*
`String toString()`	*string representation*

Complex number implementation: test client

Best practice. Begin by implementing a simple test client.

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

```
~/cos126/oop2> java-introcs Complex
a     = 3.0 + 4.0i
b     = -2.0 + 3.0i
a + b = 1.0 + 7.0i
a * b = -18.0 + 1.0i
|a|   = 5.0
```

what we expect, once the the implementation is done

- instance variables
- constructors
- instance methods
- **test client**

Complex number implementation: instance variables and constructor

Instance variables. Define data-type values.

Internal representation. Two real numbers (real and imaginary components).

each complex number has its own value (so needs its own variables)

Constructors. Create and initialize new objects.

```
public class Complex {

   private final double re;
   private final double im;

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

   ...
}
```

Complex number implementation: instance methods

Instance methods. Define data-type operations.

```
public class Complex {
   ...
   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);
   }

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

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

creates and returns a new `Complex` object

can access instance variables of any object in class by using `.` operator

could be improved (e.g., if real part is 0 or imaginary part is negative)

- instance variables
- constructors
- **instance methods**
- test client

Complex implementation

instance variables

constructor

instance methods

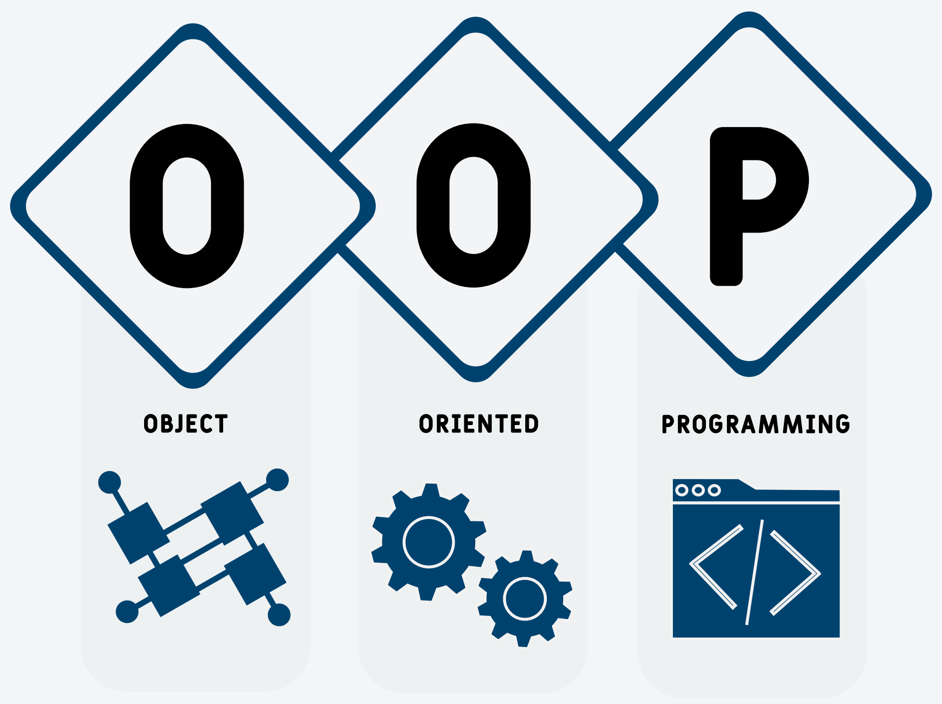
```
public class Complex {

   private final double re;
   private final double im;

   // creates a new complex object
   public Complex(double real, double imag) {
      re = real;
      im = imag;
   }

   // sum of two complex numbers
   public Complex plus(Complex b) {
      double real = re + b.re;
      double imag = im + b.im;
      return new Complex(real, imag);
   }

   // product of two complex numbers
   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);
   }

   ...

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

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

   // test client
   public static void main(String[] args) {
      Complex a = new Complex( 3.0, 4.0);
      Complex b = new Complex(-2.0, 3.0);
      StdOut.println("a     = " + a);
      StdOut.println("b     = " + b);
      StdOut.println("a + b = " + a.plus(b));
      StdOut.println("a * b = " + a.times(b));
      StdOut.println("|a|   = " + a.abs());
   }
}
```

test client

OOP summary

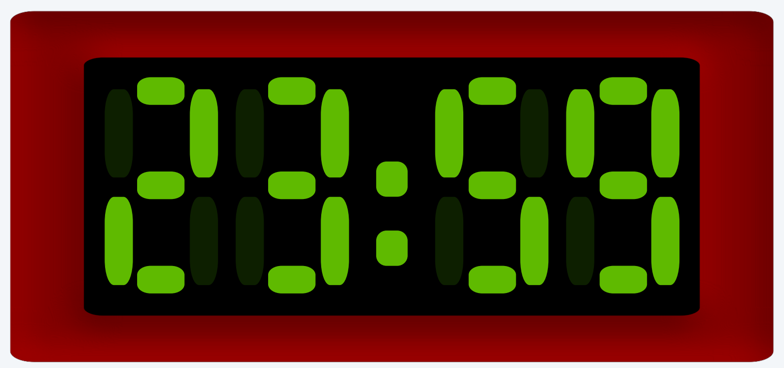
Object-oriented programming.

- Create your own data types. ← *set of values and operations on those values*
- Use data types in your programs.



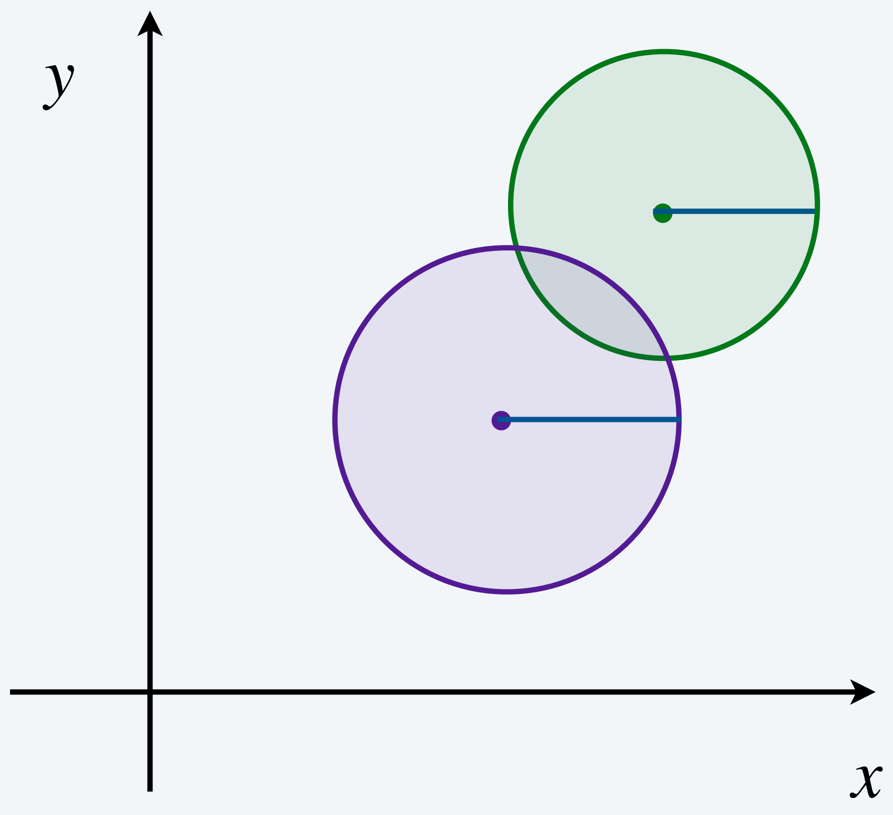
OOP helps us simulate the physical world.

- Java objects model real-world objects.
- Not always easy to make model reflect reality.
- Ex: clock, molecule, color, image, sound, genome, …



OOP helps us extend the Java language.

- Java doesn't have a data type for every conceivable application.
- Data types enable us to add our own abstractions.
- Ex: point, circle, complex number, vector, polynomial, …



Credits

image	source	license
OOP Dice	Adobe Stock	education license
3D Model of DNA Molecule	Adobe Stock	education license
Digital Clock	Wikimedia	CC BY 3.0
OOP	Adobe Stock	education license
Imaginary Number	Adobe Stock	education license