

Classes and objects



Chapter 2: Head First Java: 2nd Edition, K. Sierra, B. Bates

Overview

- Primitive types
- Creating your own data types
 - Classes
 - Objects
 - Instance variables
 - Instance methods
 - Constructors
 - Arrays of objects

Java primitive types

Java type	what it stores	examples
byte	tiny integer values -128 to 127	3 -87
short	small integer values -32768 to 32767	-3433 123
int	integer values -2,147,483,648 to 2,147,483,647	42 1234
long	big integer values -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	5454 -43984938
double	floating-point values	9.95 3.0e8
float	less precise floating-point values	9.95f 3.0e8f
boolean	truth values	true false
char	characters	'a', 'b', '!'

Primitive type limitations

- Primitive types

- Limited to **basic set of operations**

- Example: `int` has add, subtract, multiple, divide, modulo

- **Can't combine related information** together in one package

- Example: need two `double`'s to represent your Mars lander's position, another two for velocity, etc.
- Example: each point in your logging lease program had information in 4-5 different arrays

Your own types

- Create your own types

- Class

- *Blueprint* for a custom data type

- Object

- *Instance* of a class
 - May be multiple objects for a particular class blueprint
 - Objects have **a set of things they know**
 - Lander's position, velocity
 - Objects have a **set of things they can do**
 - Draw the lander
 - Update lander's position using its current velocity
 - See if the lander is out of fuel

Let's build a simple class

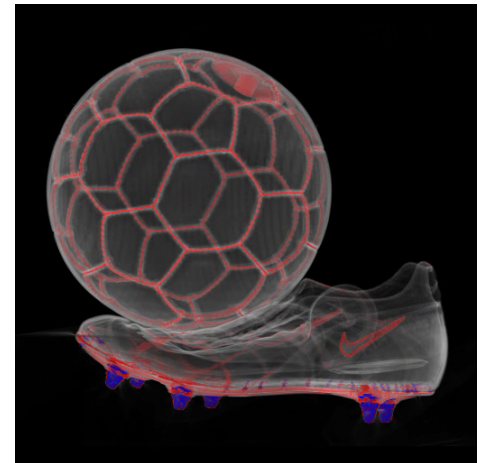
- **Goal:** represent a ball in 2D
 - **What does a ball need to know?**
 - x-coordinate
 - y-coordinate
 - radius
 - **What can a ball do?**
 - Draw itself
 - Print out its position and radius



Setting up the Ball class

- Create `Ball.java` containing `Ball` class
- Add **instance variables for what a `Ball` knows**

```
public class Ball
{
    private double posX    = 0.0;
    private double posY    = 0.0;
    private double radius  = 0.0;
}
```



Setting up the Ball class

- Create `Ball.java` containing `Ball` class
- Add **instance variables** for what a `Ball` knows

```
public class Ball
{
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;
}
```

instance variables:
variables declared
inside class but
outside any method

access modifier:

private = only methods in this class can see
and change these instance variables

Ignore the book, we declare all our instance
variables private from the get go.

Adding an instance method

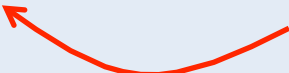
- Add instance methods for what a `Ball` can do

```
public class Ball
{
    private double posX    = 0.0;
    private double posY    = 0.0;
    private double radius  = 0.0;

    public void draw()
    {
        StdDraw.filledCircle(posX, posY, radius);
    }

    public String toString()
    {
        return "(" + posX + ", " + posY + ") r = " + radius;
    }
}
```

instance variables:
available (in scope) in any instance method of `Ball`



Adding an instance method

- Add instance methods for what a `Ball` can do

```
public class Ball
{
    private double posX = 0.0;
    private double posY = 0.0;
    private double radius = 0.0;

    public void draw()
    {
        StdDraw.filledCircle(posX, posY, radius);
    }

    public String toString()
    {
        return "(" + posX + ", " + posY + ") r = " + radius;
    }
}
```

`toString()`

Special name, called whenever object printed with

`System.out.println`

instance methods:

declared without the `static` keyword

Let's try out our new class!

- Instantiating objects

- Like arrays, we must declare and create using `new`

```
public class BallClient
{
    public static void main(String [] args)
    {
        Ball big    = new Ball();
        Ball small = new Ball();

        big.draw();
        small.draw();

        System.out.println("big: " + big);
        System.out.println("small: " + small);
    }
}
```

Let's try out our new class!

- Instantiating objects

- Like arrays, we must declare and create using `new`

```
public class BallClient
{
    public static void main(String [] args)
    {
        Ball big    = new Ball();
        Ball small  = new Ball();

        big.draw();
        small.draw();

        System.out.println("big: " + big);
        System.out.println("small: " + small);
    }
}
```

"Build me a Ball object, I'm not sending you any input about how to do it."

Let's try out our new class!

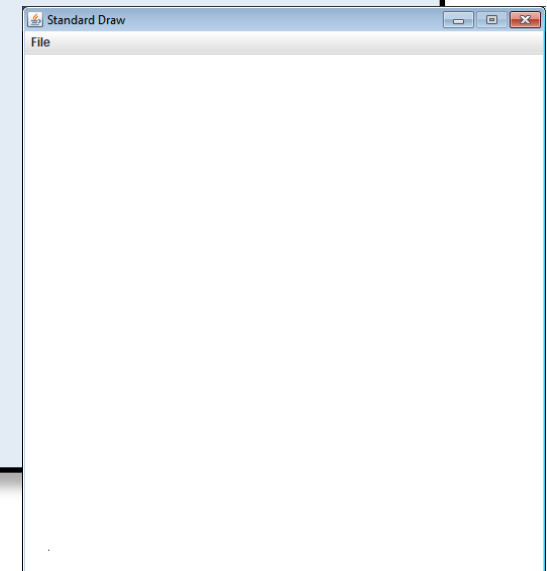
- Instantiating objects

- Like arrays, we must declare and create using `new`

```
public class BallClient
{
    public static void main(String [] args)
    {
        Ball big    = new Ball();
        Ball small  = new Ball();

        big.draw();
        small.draw();

        System.out.println("big: " + big);
        System.out.println("small: " + small);
    }
}
```



```
% java BallClient
big: (0.0, 0.0) r = 0.0
small: (0.0, 0.0) r = 0.0
```

Hello constructors

- Add a **constructor** method, **sets instance vars**

```
public class Ball
{
    private double posX      = 0.0;
    private double posY      = 0.0;
    private double radius    = 0.0;

    public Ball(double x, double y, double r)
    {
        posX      = x;
        posY      = y;
        radius    = r;
    }

    public void draw()
    {
        StdDraw.filledCircle(posX, posY, radius);
    }

    public String toString()
    {
        return "(" + posX + ", " + posY + ") r = " + radius;
    }
}
```

Hello constructors

- Add a **constructor** method, sets instance vars

```
public class Ball
{
    private double posX      = 0.0;
    private double posY      = 0.0;
    private double radius    = 0.0;

    public Ball(double x, double y, double r)
    {
        posX      = x;
        posY      = y;
        radius    = r;
    }

    public void draw()
    {
        StdDraw.filledCircle(posX, posY, radius);
    }

    public String toString()
    {
        return "(" + posX + ", " + posY + ") r = " + radius;
    }
}
```

constructor:
No return type.
Method name same as class.
These are requirements!

BallClient take two

- **Constructor called** when we `new` objects

```
public class BallClient
{
    public static void main(String [] args)
    {
        Ball big    = new Ball(0.7, 0.7, 0.2);
        Ball small  = new Ball(0.1, 0.5, 0.1);

        big.draw();
        small.draw();

        System.out.println("big: " + big);
        System.out.println("small: " + small);
    }
}
```


BallClient take two

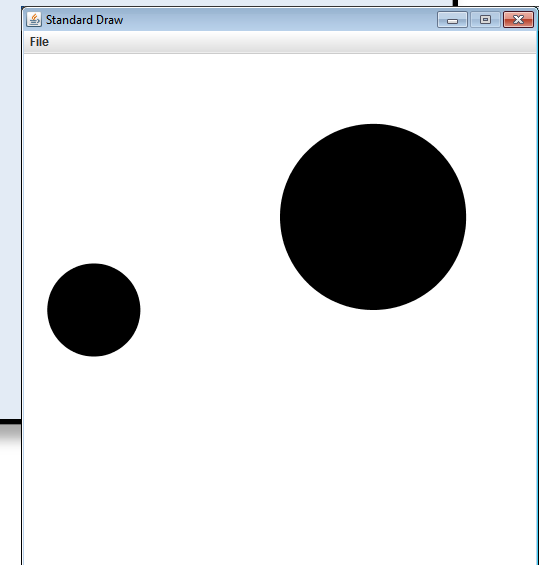
- **Constructor called** when we `new` objects

```
public class BallClient
{
    public static void main(String [] args)
    {
        Ball big    = new Ball(0.7, 0.7, 0.2);
        Ball small  = new Ball(0.1, 0.5, 0.1);

        big.draw();
        small.draw();

        System.out.println("big: " + big);
        System.out.println("small: " + small);
    }
}
```

```
% java BallClient
big: (0.1, 0.5) r = 0.1
small: (0.7, 0.7) r = 0.2
```



Colored balls

- **Goal:** make each Ball object have a color specified by an red-green-blue (RGB) value
- Call `StdDraw.setPenColor()` in `draw()`
 - Create a new `Color` object for a given RGB value
 - `Color` is a class in the Java API
 - Default color for our Ball objects: [mauve](#)



Ball in living color

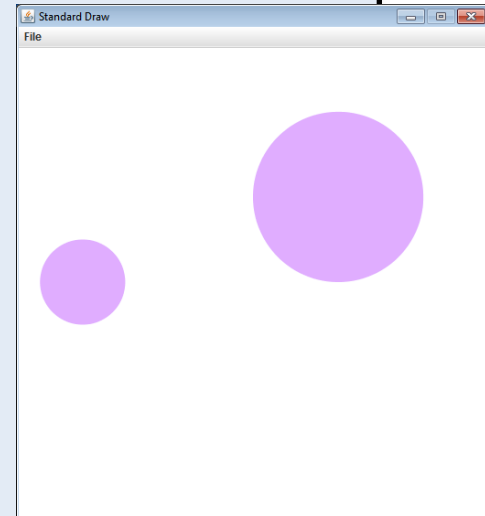
```
import java.awt.*;

public class Ball
{
    private double posX    = 0.0;
    private double posY    = 0.0;
    private double radius  = 0.0;
    private Color  color   = new Color(0.88f, 0.68f, 1.0f);

    public Ball(double x, double y, double r)
    {
        posX    = x;
        posY    = y;
        radius  = r;
    }

    public void draw()
    {
        StdDraw.setPenColor(color);
        StdDraw.filledCircle(posX, posY, radius);
    }

    ...
}
```



Allowing clients to change color

```
import java.awt.*;

public class Ball
{
    private double posX    = 0.0;
    private double posY    = 0.0;
    private double radius  = 0.0;
    private Color  color   = new Color(0.88f, 0.68f, 1.0f);

    public Ball(double x, double y, double r)
    {
        posX    = x;
        posY    = y;
        radius  = r;
    }

    public void setColor(double r, double g, double b)
    {
        color = new Color((float) r, (float) g, (float) b);
    }

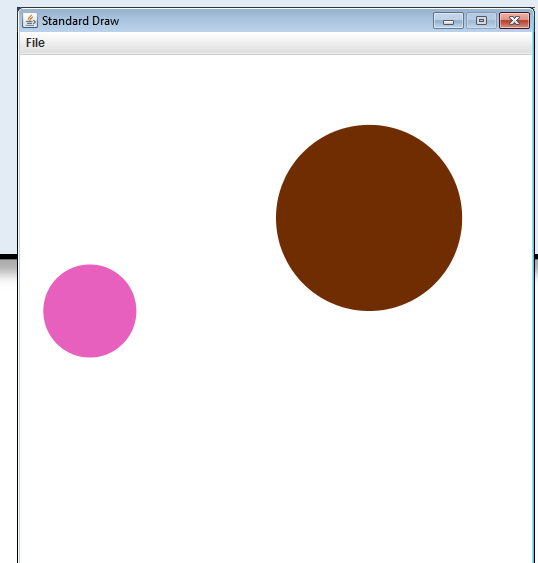
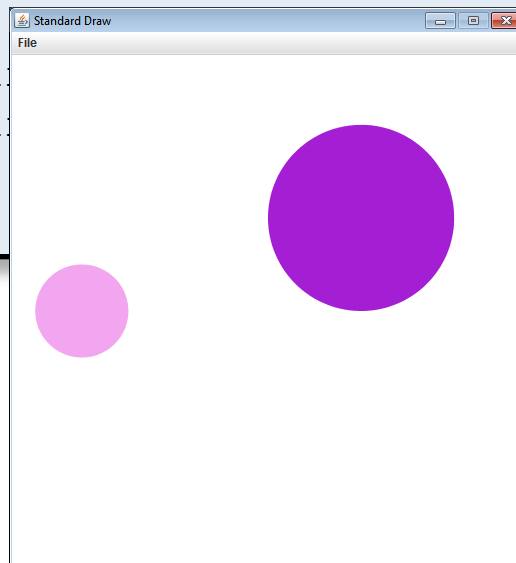
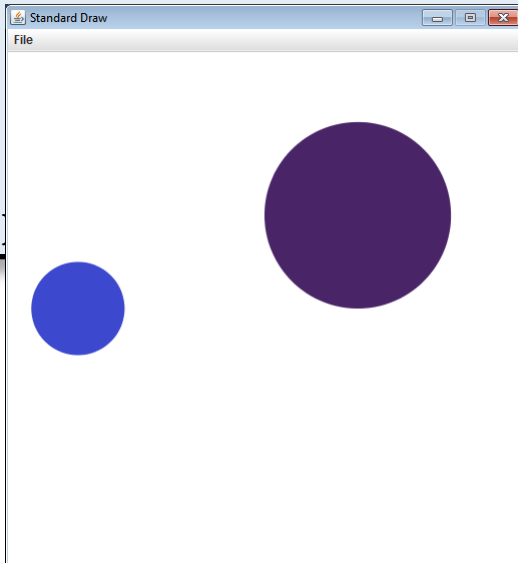
    ...
}
```

Client setting random color

```
public class BallClient
{
    public static void main(String [] args)
    {
        Ball big    = new Ball(0.7, 0.7, 0.2);
        Ball small  = new Ball(0.1, 0.5, 0.1);

        big.setColor(Math.random(), Math.random(), Math.random());
        small.setColor(Math.random(), Math.random(), Math.random());

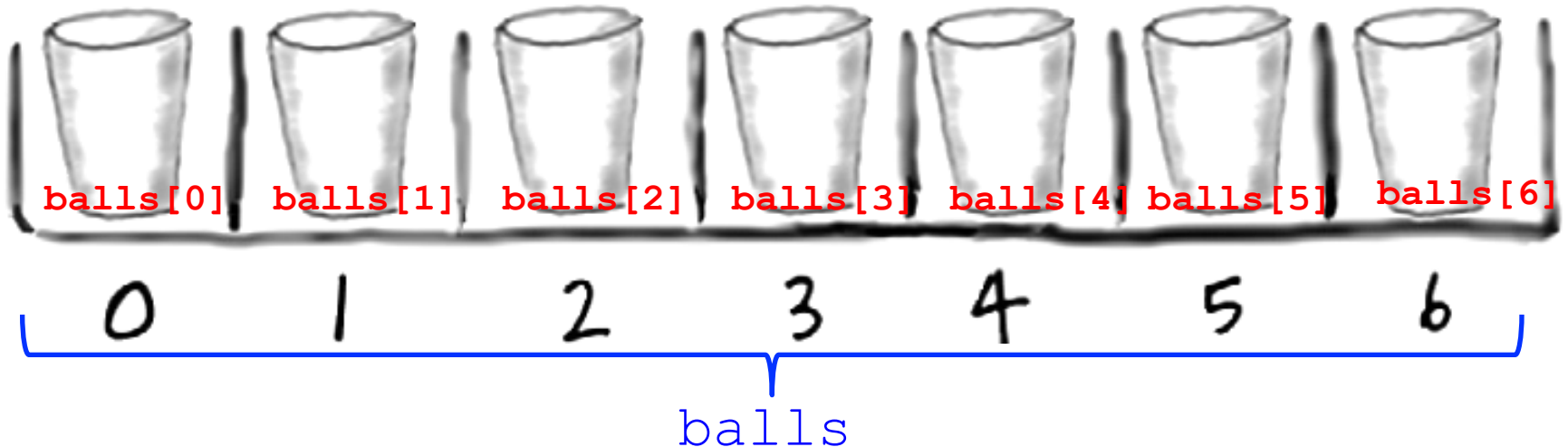
        big.draw();
        small.draw();
    }
}
```



Creating lots of balls

- We can have an **array of objects**
- **Step 1:** create an array to hold `Ball` objects

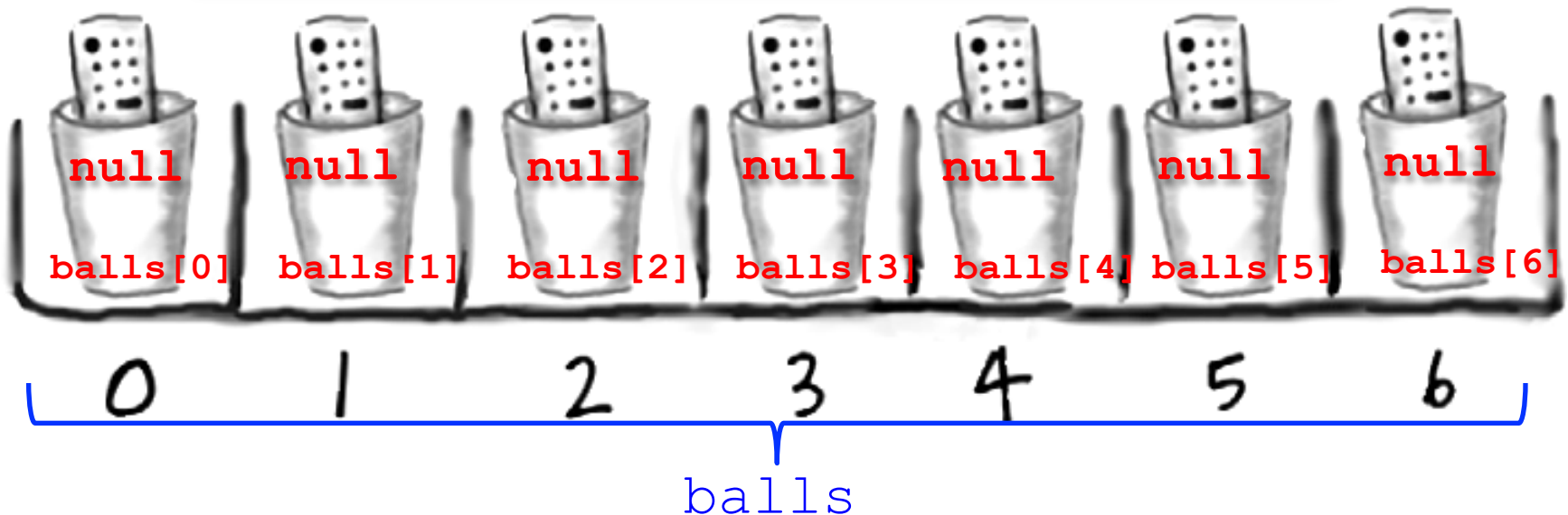
```
Ball [] balls = new Ball[7];
```



The value `null`

- What is in each location of the array?
 - Special value `null`
 - Default value for reference types (non-primitives)
 - Unprogrammed remote control

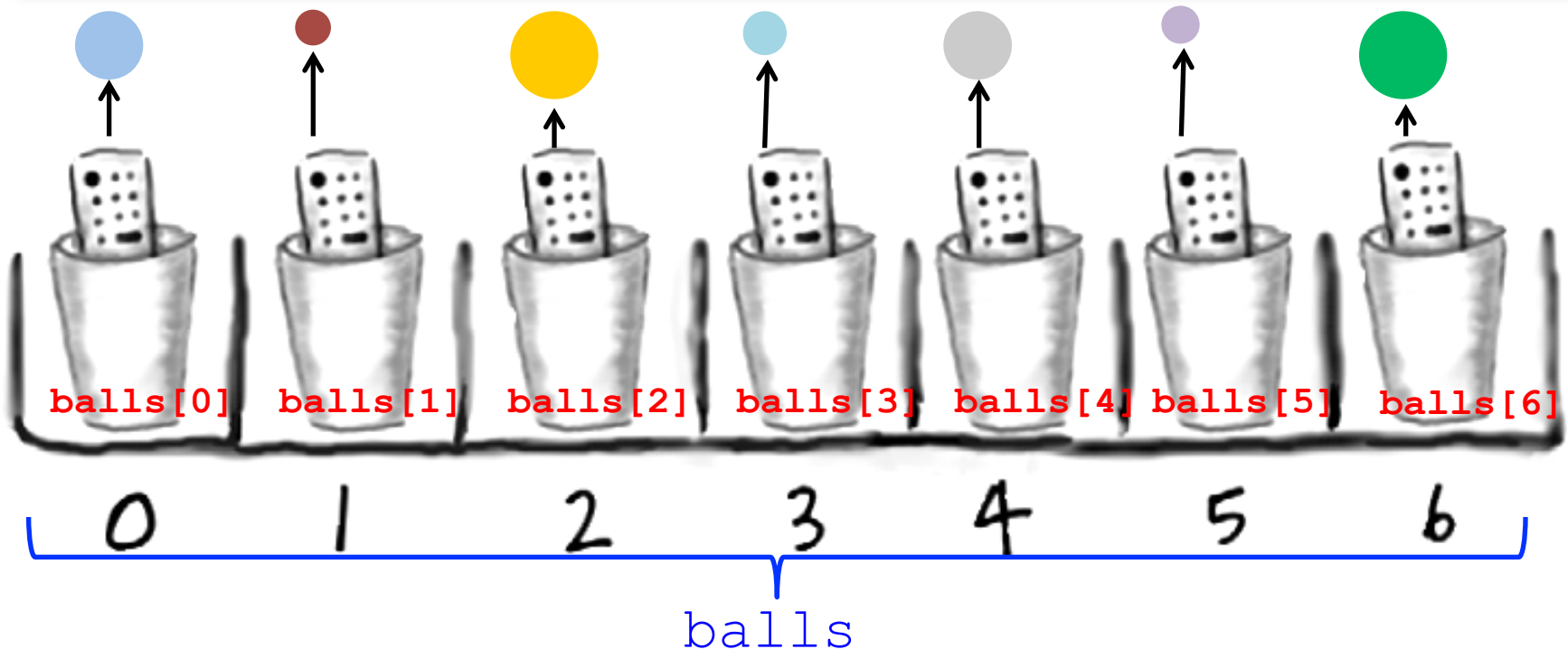
```
Ball [] balls = new Ball[7];
```



Creating all the Ball objects

- Each array location needs a new object

```
Ball [] balls = new Ball[7];  
for (int i = 0; i < balls.length; i++)  
{  
    balls[i] = new Ball(Math.random(), Math.random(),  
                        Math.random() * 0.2);  
    balls[i].setColor(Math.random(), Math.random(), Math.random());  
}
```



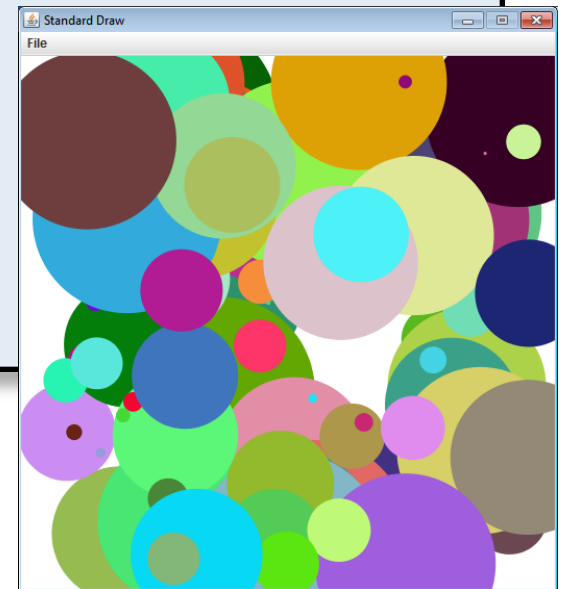
Client to draw lots of Ball objects

```
public class BallClientDeluxe
{
    public static void main(String[] args)
    {
        Ball [] balls = new Ball[Integer.parseInt(args[0])];
        for (int i = 0; i < balls.length; i++)
        {
            balls[i] = new Ball(Math.random(),
                                Math.random(),
                                Math.random() * 0.2);
            balls[i].setColor(Math.random(),
                              Math.random(),
                              Math.random());
            balls[i].draw();
        }
    }
}
```

Client to draw lots of Ball objects

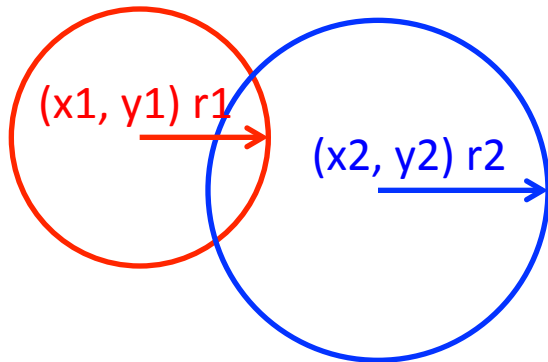
```
public class BallClientDeluxe
{
    public static void main(String[] args)
    {
        Ball [] balls = new Ball[Integer.parseInt(args[0])];
        for (int i = 0; i < balls.length; i++)
        {
            balls[i] = new Ball(Math.random(),
                                Math.random(),
                                Math.random() * 0.2);
            balls[i].setColor(Math.random(),
                              Math.random(),
                              Math.random());
            balls[i].draw();
        }
    }
}
```

```
% java BallClientDeluxe 100
```



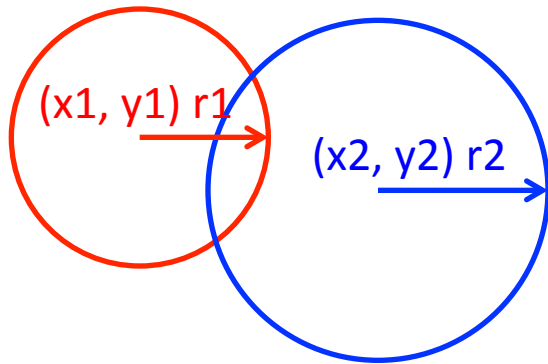
Overlap detection

- **Goal:** draw many Ball objects without overlap
 - When do two balls overlap?



Overlap detection

- **Goal:** draw many Ball objects without overlap
 - When do two balls overlap?



Euclidean distance between centers:

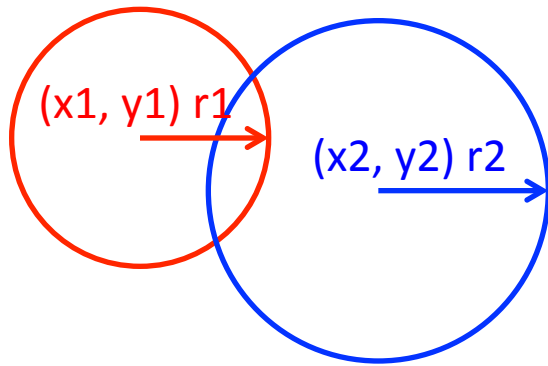
$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Balls overlap if:

$$d < (r_1 + r_2)$$

Implementing overlap detection

- Overlap detection is **something a Ball can do**
 - We can **add a method** to `Ball` class for this!



Euclidean distance between centers:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Balls overlap if:

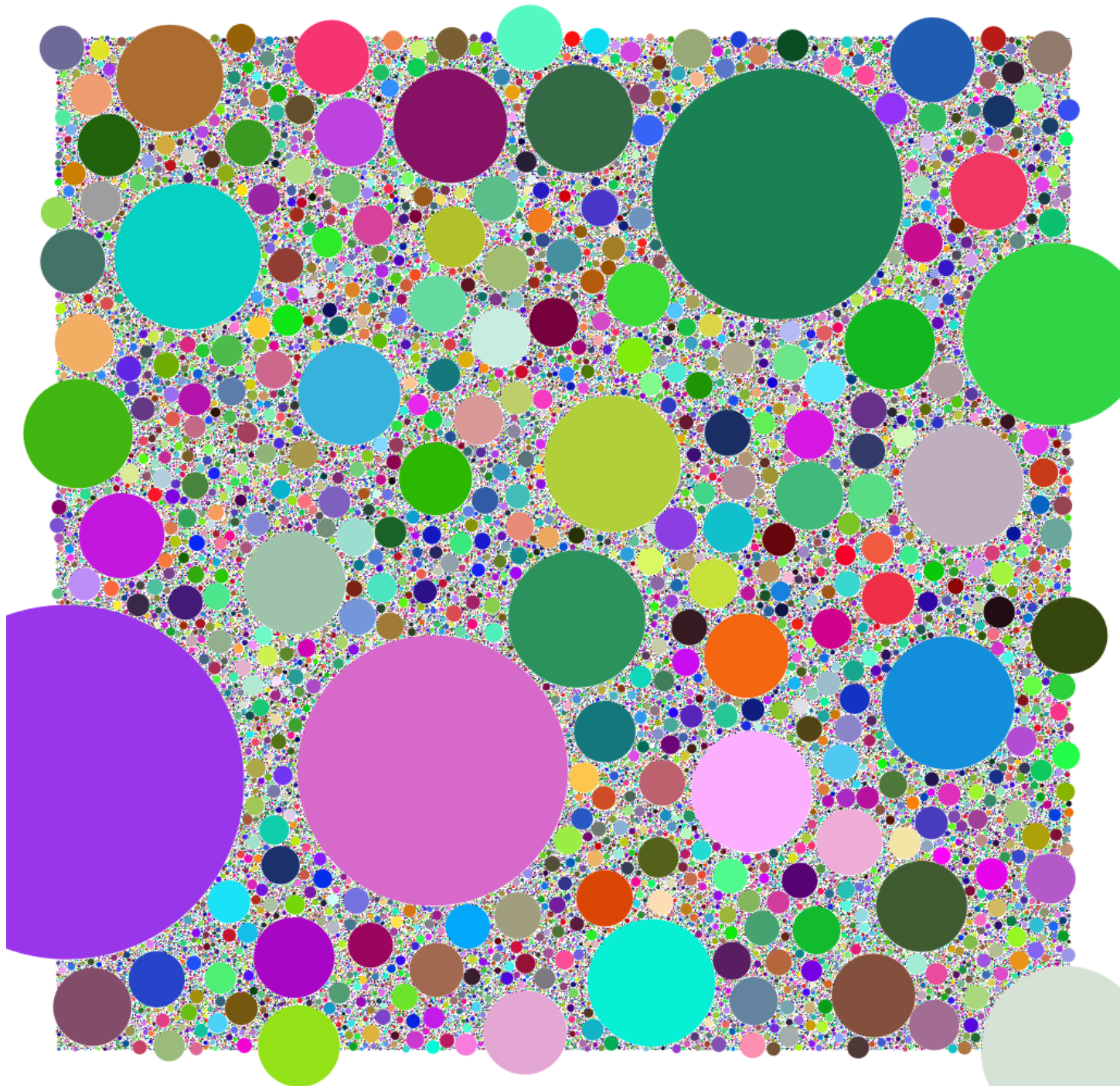
$$d < (r_1 + r_2)$$

```
public boolean overlap(Ball other)
{
    double deltaX = posX - other.posX;
    double deltaY = posY - other.posY;
    double d = Math.sqrt(deltaX * deltaX + deltaY * deltaY);
    if (d < (radius + other.radius))
        return true;
    return false;
}
```

BallClientSuperDeluxe

```
public class BallClientSuperDeluxe
{
    public static void main(String[] args)
    {
        Ball [] balls = new Ball[Integer.parseInt(args[0])];
        for (int i = 0; i < balls.length; i++)
        {
            boolean overlap = false;
            do
            {
                balls[i] = new Ball(Math.random(),
                                    Math.random(),
                                    Math.random() * 0.2);

                int j = 0;
                overlap = false;
                while ((j < i) && (!overlap))
                {
                    overlap = balls[i].overlap(balls[j]);
                    j++;
                }
            } while (overlap);
            balls[i].setColor(Math.random(),
                              Math.random(),
                              Math.random());
            balls[i].draw();
        }
    }
}
```



Summary

- Creating your own data types
 - Object-oriented programming (OOP)
 - Design classes encapsulating:
 - What objects know
 - What objects can do
 - Prevalent concept in most modern programming languages