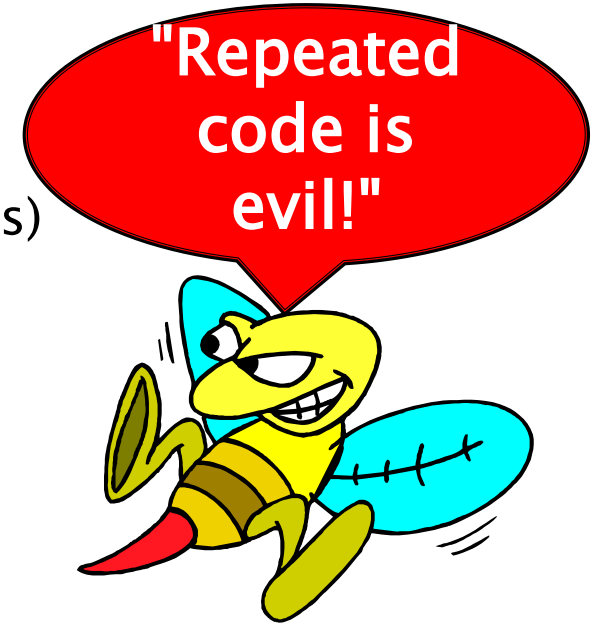
MontanaTech
THE UNIVERSITY OF MONTANA

Fundamentals of Computer Science II: Review of Objects

Michele Van Dyne
Museum 204B
mvandyne@mtech.edu
http://katie.mtech.edu/classes/csci136

Review of Objects: Outline

- Static Methods (and Methods in General)
- Creating Data Types
  - Classes and Objects
  - Arrays of objects
- Object Oriented Design and Programming
  - State: Instance variables
  - Behavior: Instance methods
  - Constructors
  - Encapsulation
    - Accessors (Getters) and Mutators (Setters)
  - Other Methods
    - toString
    - equals
- API



Static Methods

- Static "helper" methods we've used:

```
System.out.println("Hello world");
StdDraw.setPenColor(StdDraw.GRAY);
int num    = Integer.parseInt(args[0]);
double r   = Double.parseDouble(args[1]);
int x      = StdIn.readInt();
double rand = Math.random();
double v   = Math.pow(10.0, -2.3582);
StdDraw.setXscale(0.0, 10.0);
```

Methods in General

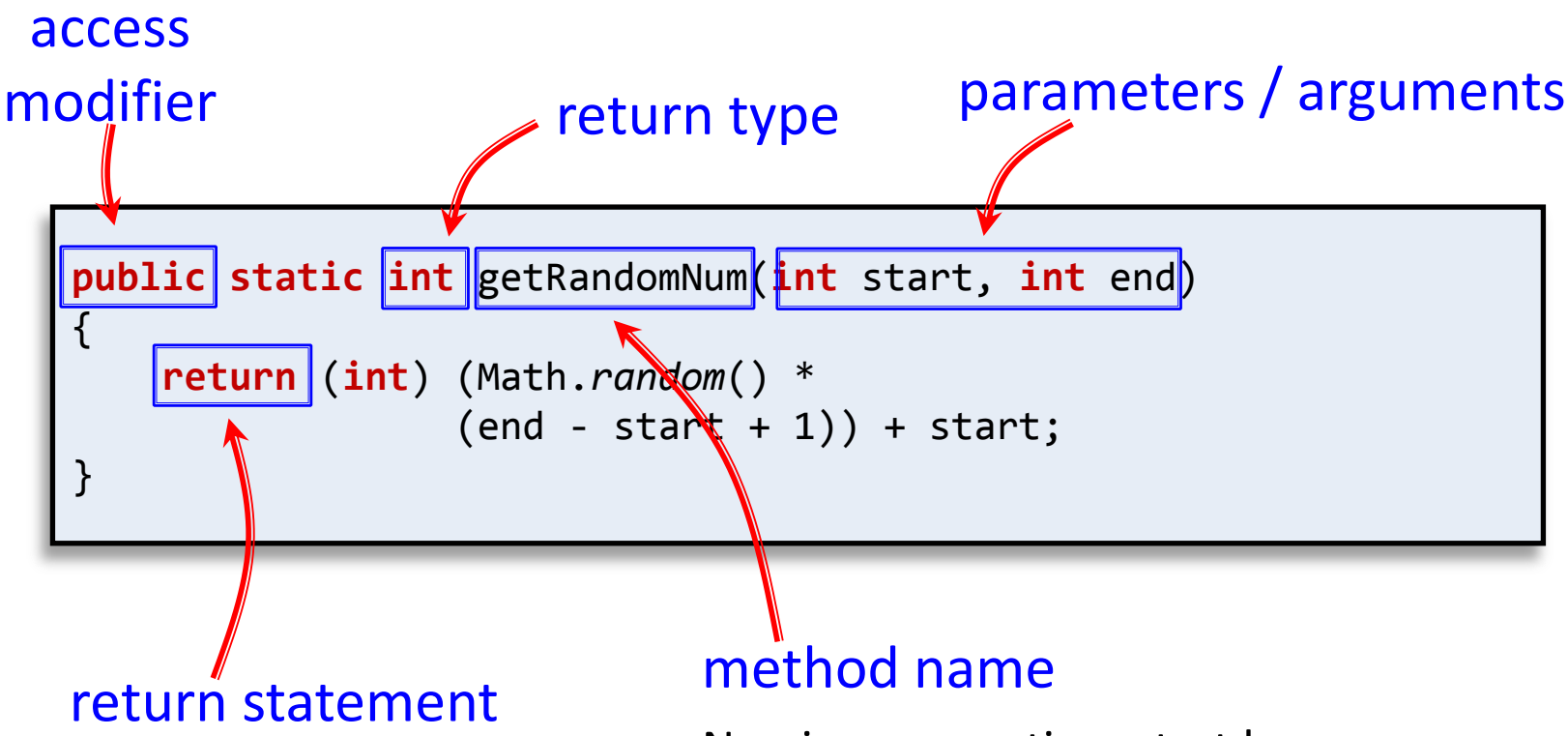
- Methods:
  - Like a mathematical function
    - Given some inputs, produce an output value
  - Methods allows building modular programs
    - Reuse code, only invent the wheel once
  - When a method is called:
    - Control jumps to the method code
    - Argument passed to method copied to parameter variables used in method
    - Method executes and (optionally) returns a value
    - Execution returns to calling code

Methods: Flow of Control

```
public class MethodJumping
{
    public static void printWorld()
    {
        System.out.print("world");
    }
    public static int addNums(int num1, int num2)
    {
        int result = num1;
        result = num1 + num2;
        return result;
    }
    public static void main(String [] args)
    {
        System.out.print("Hello");
        System.out.print(" ");
        printWorld();
        System.out.print(", 1 + 2 = ");
        int a = addNums(1, 2);
        System.out.println(a);
    }
}
```

```
% java MethodJumping
Hello world, 1 + 2 = 3
```

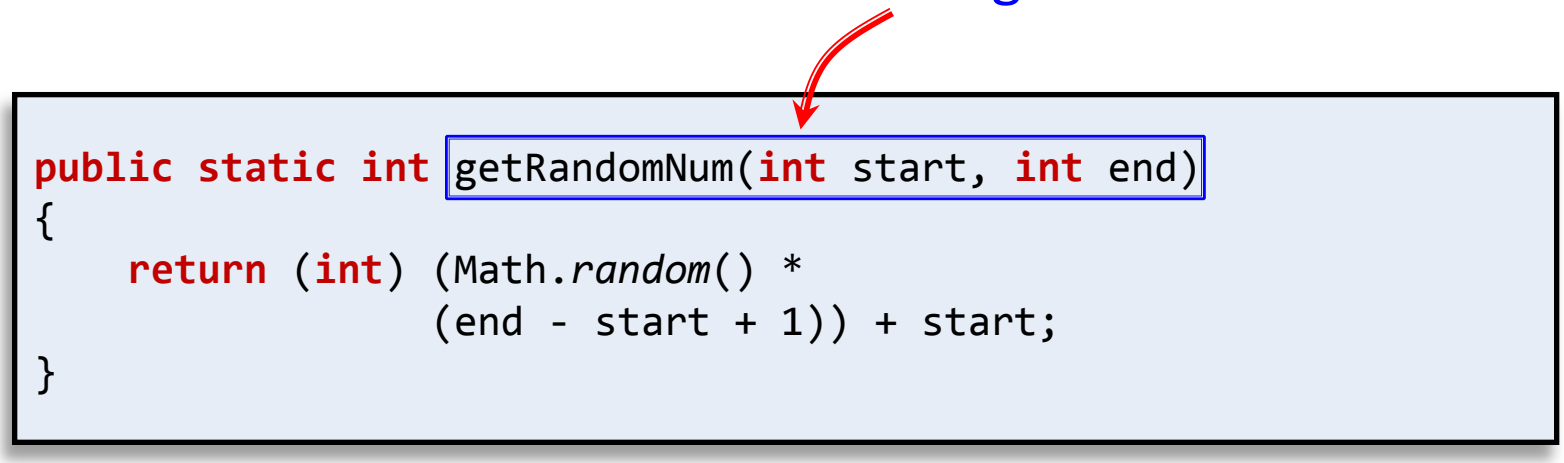
Methods: Terminology



Naming convention: start lowercase, uppercase each new word

Methods: Signature

- Signature: a method's name plus the number and type of its parameters
  - Note: does NOT include the return type!



Methods: Pass by Value

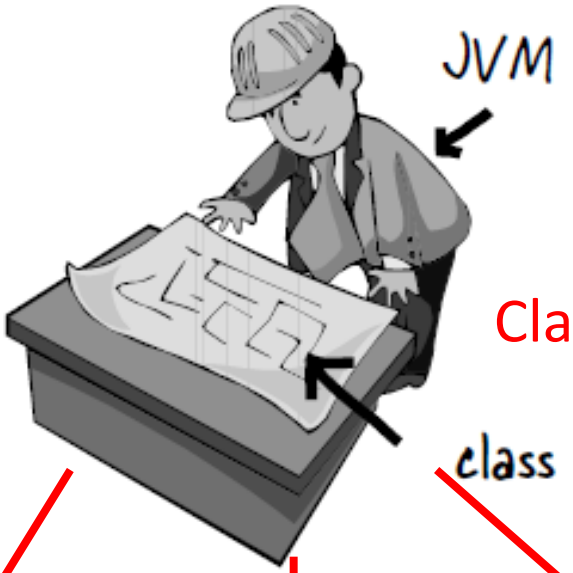
- Java passes parameters by value (by copy)
  - Changes to primitive type parameters do not persist after method returns
    - Primitive types: `int`, `double`, `char`, `long`, `boolean`

```
public static int sum(int a, int b)
{
   int result = a + b;
   a = 0;
   b = 0;
   return result;
}
```

```
% java PassByVal
sum = 5
c = 2
d = 3
```

```
int c = 2;
int d = 3;
System.out.println("sum = " + sum(c, d));
System.out.println("c = " + c);
System.out.println("d = " + d);
```

Classes and objects



Class = object blueprint

0.1,
0.5
r=0.1

0.5, 0.3
r=0.2

0.9,
0.6
r=0.15

Objects = different instances of a class

Creating Data Types (Classes)

- Object Oriented Programming (OOP)
  - Create your own data types
  - Use them in your programs
- Objects
  - Holds a data type value
  - Variable name refers to object

Remember – a data type is a set of legal values and the operations defined on those values

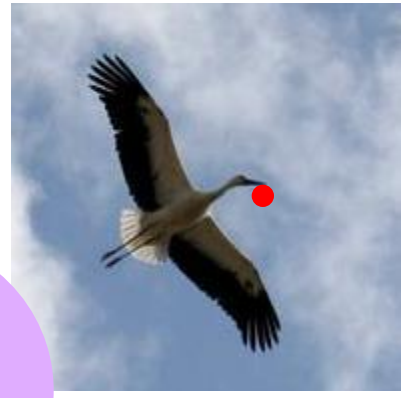
Data type	Set of values	Example operations
`Color`	24 bits	get red component, brighten
`Picture`	2D array of colors	get/set color of pixel (i, j)
`String`	sequence of characters	length, substring, compare

Object Oriented Programming

- Procedural programming [verb-oriented]
  - Tell the computer to do this
  - Tell the computer to do that
- OOP philosophy
  - Software simulation of real world
  - We know (approximately) how the real world works
  - Design software to model the real world
- Objected oriented programming (OOP) [noun-oriented]
  - Programming paradigm based on data types
  - Identify: objects that are part of problem domain or solution
    - Objects are distinguishable from each other (references)
  - State: objects know things (instance variables)
  - Behavior: objects do things (methods)

Hey objects, where did you come from?

"Dude, don't you know where objects come from?!? The object stork totally dropped us off."



0.1,
0.5
r=0.1

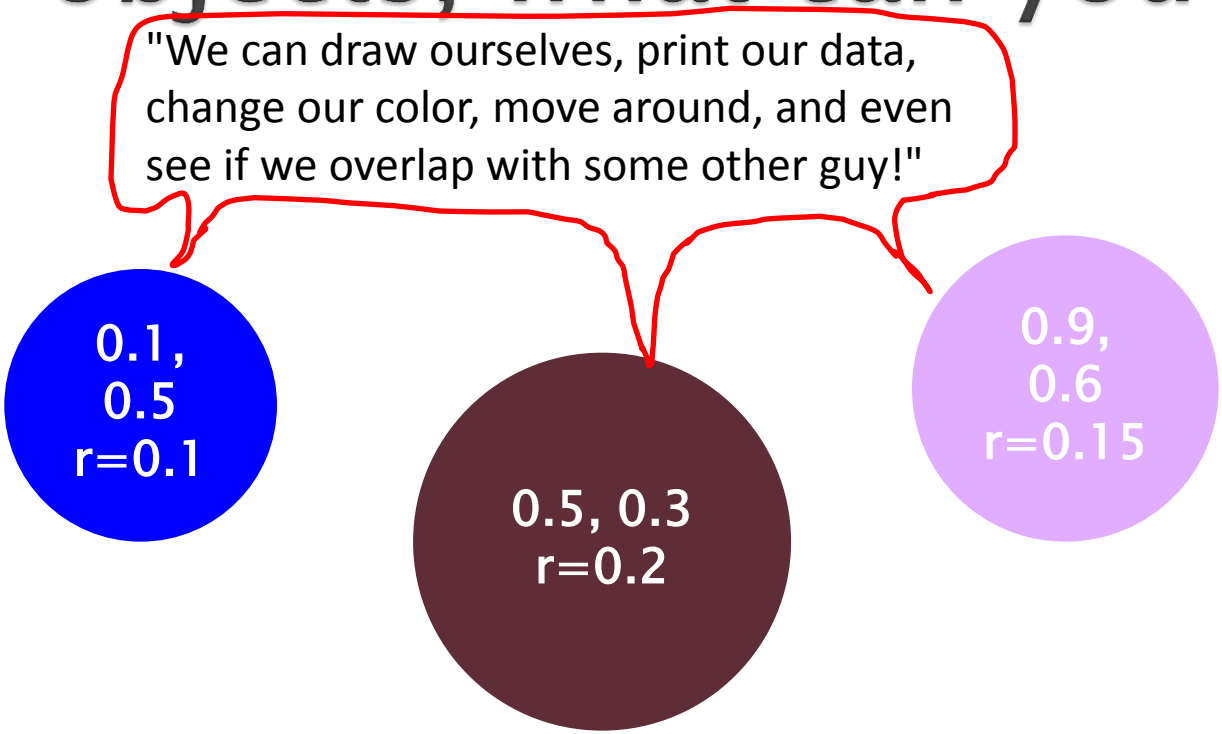
0.5, 0.3
r=0.2

0.9,
0.6
r=0.15

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

The object stork = Constructor Method

Hey objects, what can you do?

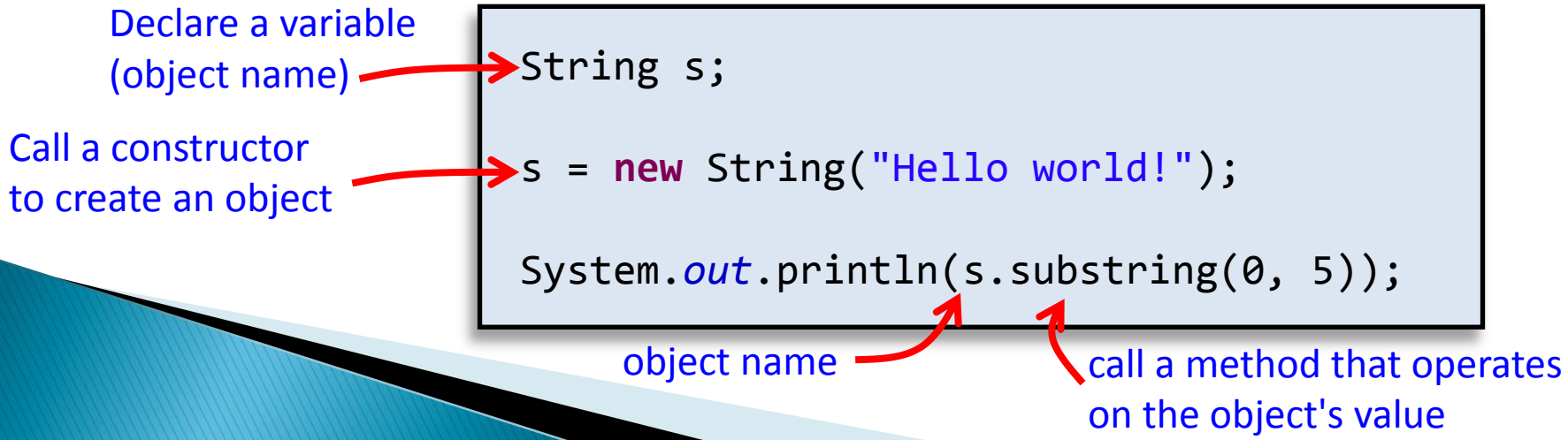



```
public void draw()                                      {...}
public String toString()                                {...}
public void setColor(double r, double g, double b)      {...}
public void move(double deltaX, double deltaY)          {...}
public boolean overlap(Ball other)                      {...}
```

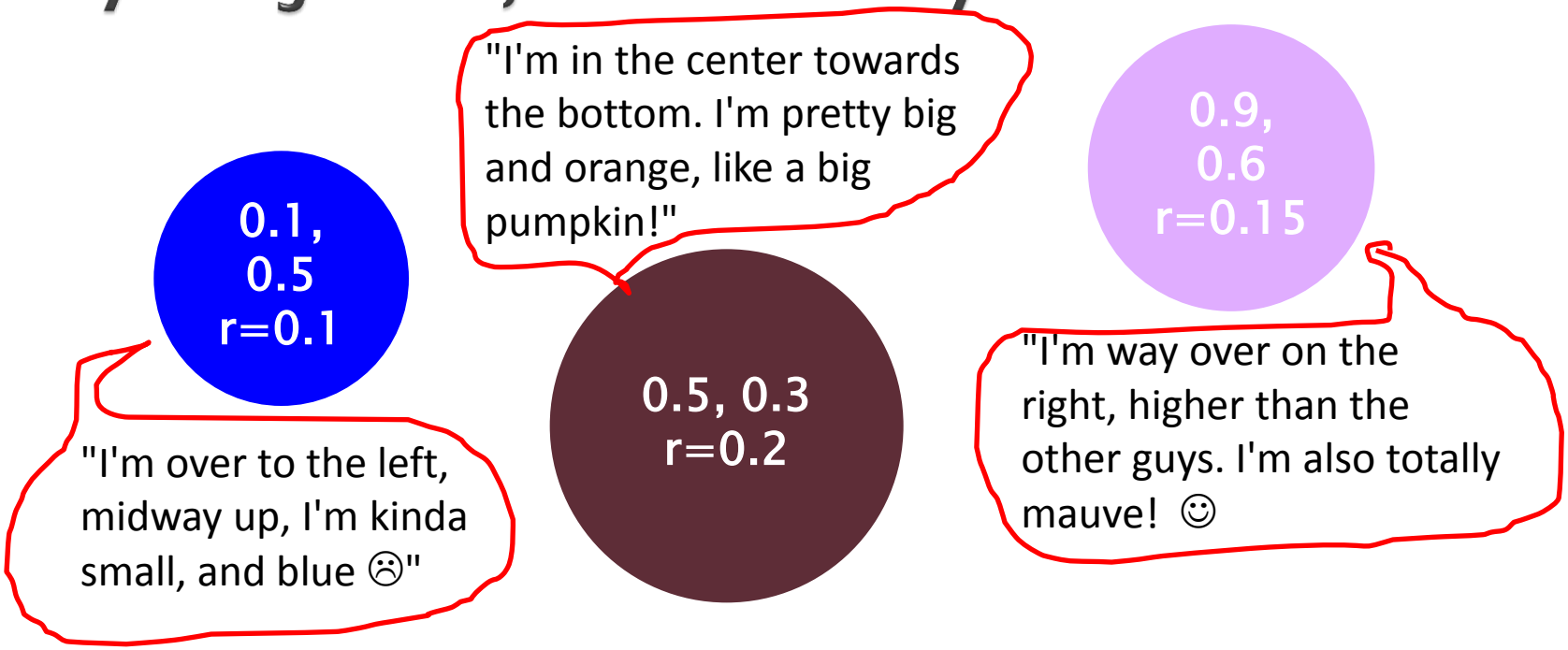
What an object can do = Instance Methods = Behavior

Constructors and Methods

- To construct a new object:
  - Use keyword new (to invoke the constructor)
  - Use name of data type (to specify which object type)
- To apply an operation:
  - Use name of object (to specify which object)
  - Use dot operator (to invoke a method)
  - Use name of the method (to specify operation)



Hey objects, what do you know?




```
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);
}
```

What an object knows = Instance Variables = State

Data encapsulation

- Data type (aka class)
  - "Set of values and operations on those values"
  - e.g. `int`, `String`, `Charge`, `Picture`, `Enemy`, `Player`
- Encapsulated data type
  - Hide internal representation of data type.
- Separate implementation from design specification
  - Class provides data representation & code for operations
  - Client uses data type as black box
  - API specifies contract between client and class
- Bottom line:
  - You don't need to know how a data type is implemented in order to use it

Access modifiers

- Access modifier
  - All instance variables and methods have one:
    - `public` – everybody can see/use
    - `private` – only class can see/use
    - `default` – everybody in package (stay tuned), what you get if you don't specify an access modifier!
    - `protected` – everybody in package and subclasses (stay tuned) outside package
  - Normally:
    - Instance variables are `private`
    - API methods the world needs are `public`
    - Helper methods used only inside the class are `private`

Getters and setters

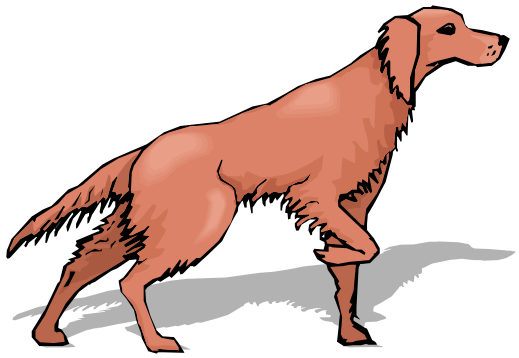
- Encapsulation does have a price
  - If clients need access to instance var, must create:
    - getter methods – "get" value of an instance var
    - setter methods – "set" value of an instance var

```
public double getPosX()
{
   return posX;
}
```

Getter method.
Also know as an ***accessor*** method.

```
public void setPosX(double x)
{
   posX = x;
}
```

Setter method.
Also know as a ***mutator*** method.



Immutability: Pros and Cons

- Immutable data type
  - Object's value cannot change once constructed
- Advantages
  - Avoid aliasing bugs
  - Makes program easier to debug
  - Limits scope of code that can change values
  - Pass objects around without worrying about modification
- Disadvantage
  - New object must be created for every value

Final access modifier

- Final
  - Declaring variable **final** means that you can assign value only once, in initializer or constructor

```
public class Counter
{
    private final String name;
    private int count;
    ...
}
```

This value doesn't change once the object is constructed

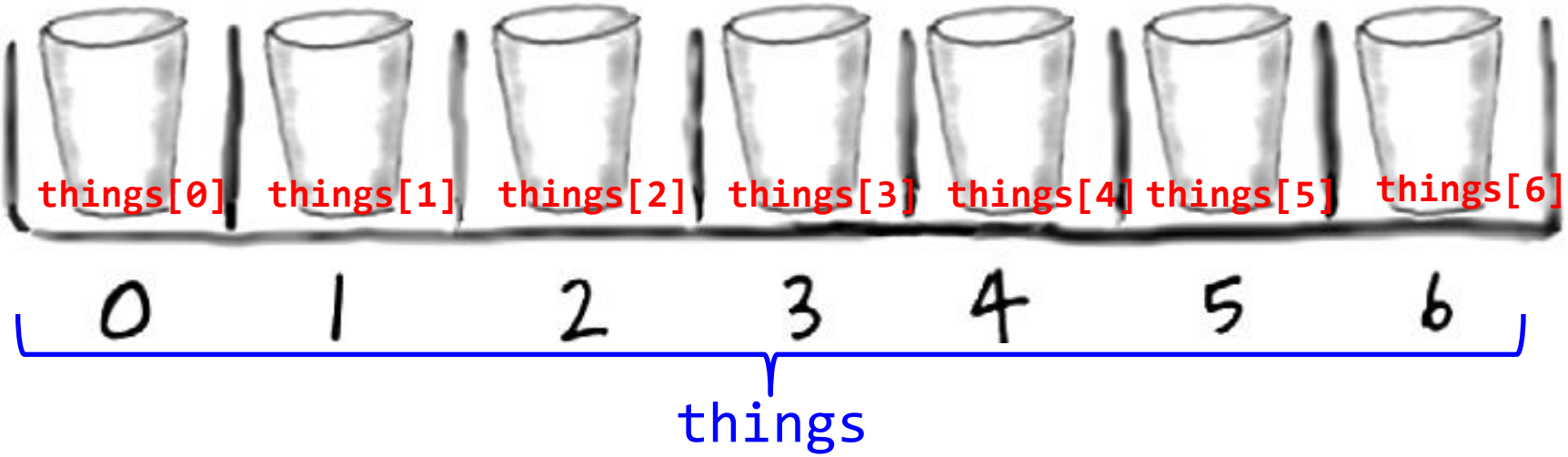
This value can change in instance methods

- Advantages
  - Helps enforce **immutability**
  - Prevents accidental changes
  - Makes program easier to debug
  - Documents that the value cannot not change

Arrays of Objects

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

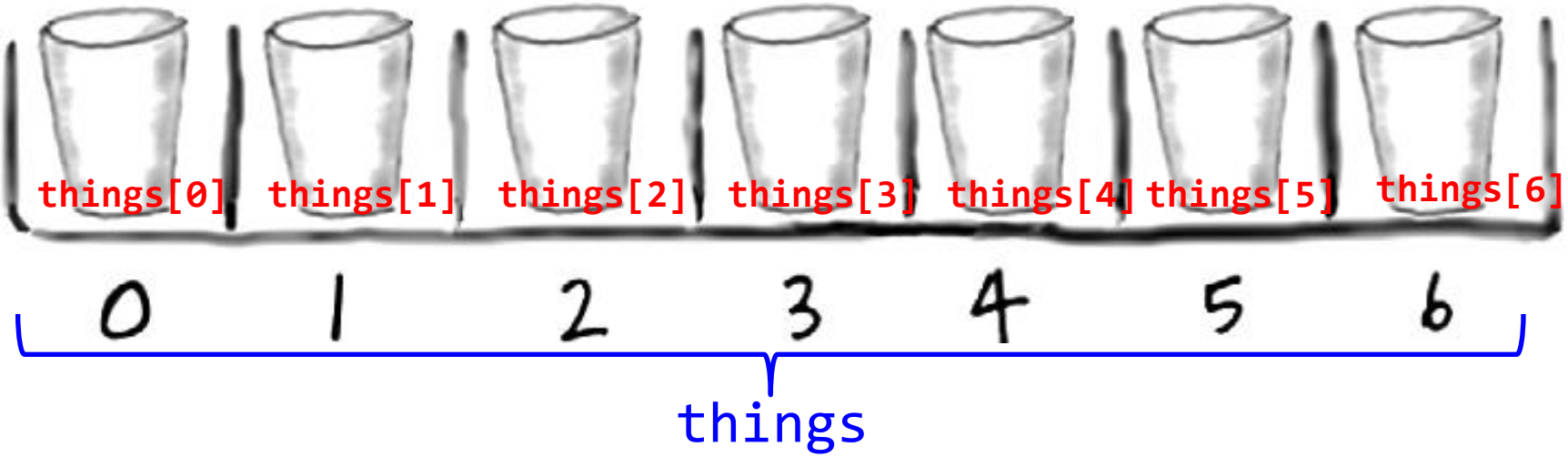
```
Thing[] things = new Thing[7];
```



Arrays of Objects

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

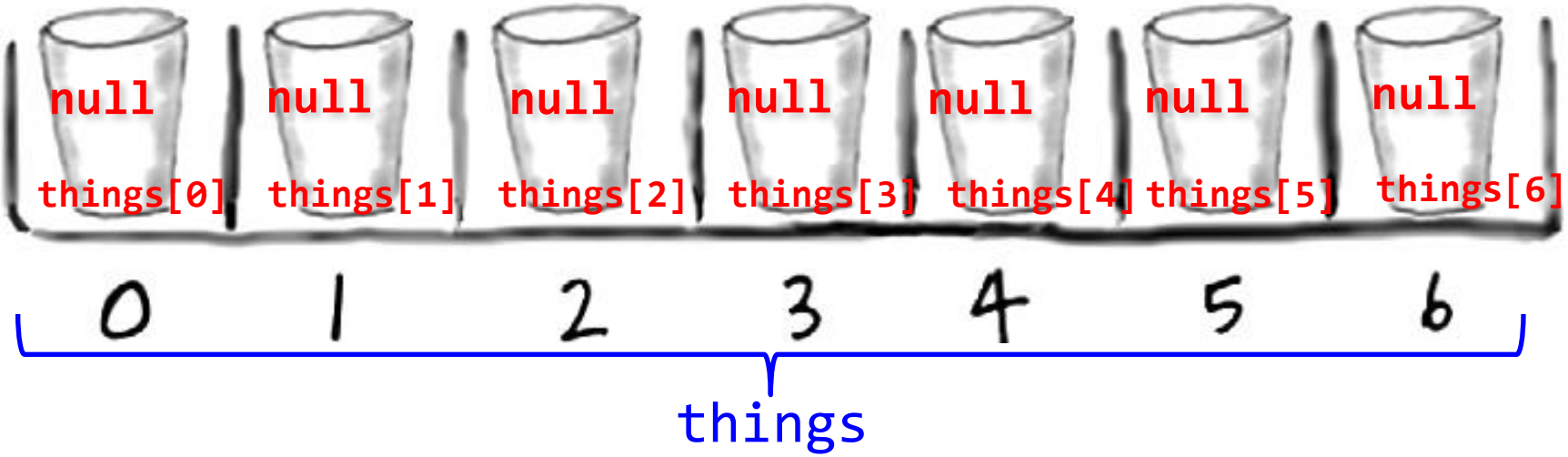
```
Thing[] things = new Thing[7];
```



Arrays of Objects: Null

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

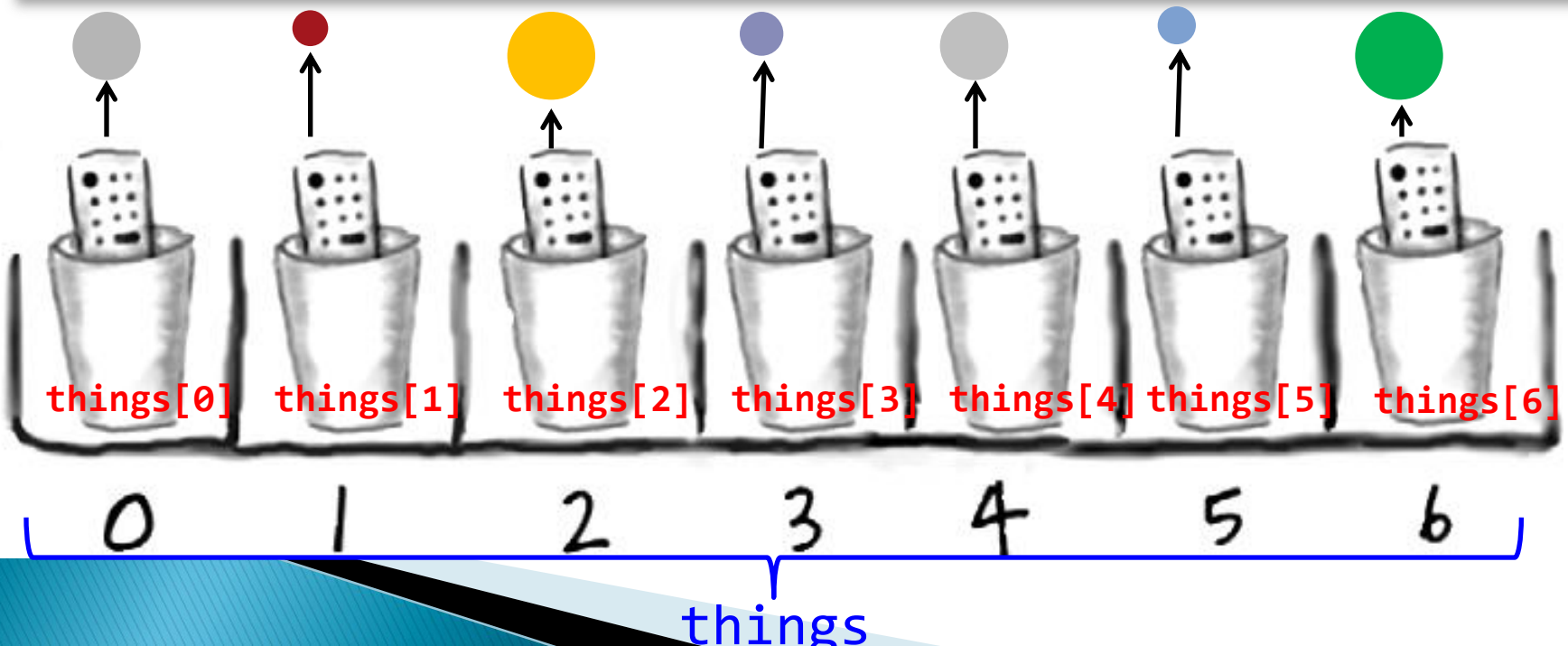
```
Thing[] things = new Thing[7];
```



Creating all the Objects

- Each array location needs a new object

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



this to Refer to Instance Variables

- this
  - Refers to the instance of the object running the method
  - Use instance variable instead of local variable

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

   public Thing(double posX, double posY, double
radius)
   {
      this.posX   = posX;
      this.posY   = posY;
      this.radius = radius;
   }
   ...
}
```

This works just fine. Using `this` allows you to have the same parameter variables names as your instance variables (if you want).

Other Methods to Include

- toString
- equals

boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
String	toString() Returns a string representation of the object.

API

protected Object	clone() Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
protected void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	getClass() Returns the runtime class of this Object.
int	hashCode() Returns a hash code value for the object.
void	notify() Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll() Wakes up all threads that are waiting on this object's monitor.
String	toString() Returns a string representation of the object.
void	wait() Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
void	wait(long timeout) Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
void	wait(long timeout, int nanos) Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Review of Objects: Summary

- Static Methods (and Methods in General)
- Creating Data Types
  - Classes and Objects
  - Arrays of objects
- Object Oriented Design and Programming
  - State: Instance variables
  - Behavior: Instance methods
  - Constructors
  - Encapsulation
    - Accessors (Getters) and Mutators (Setters)
  - Other Methods
    - toString
    - equals
- API