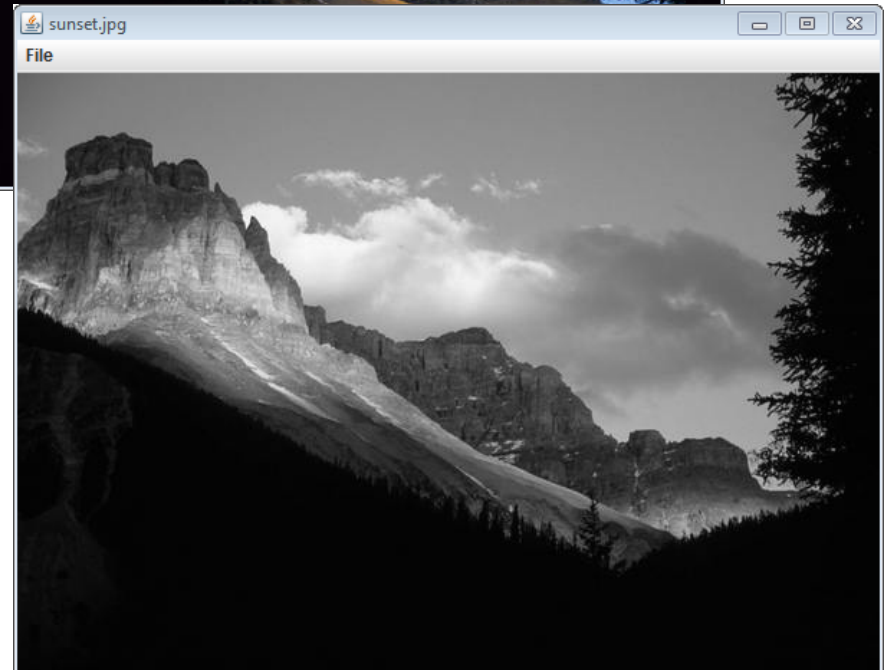
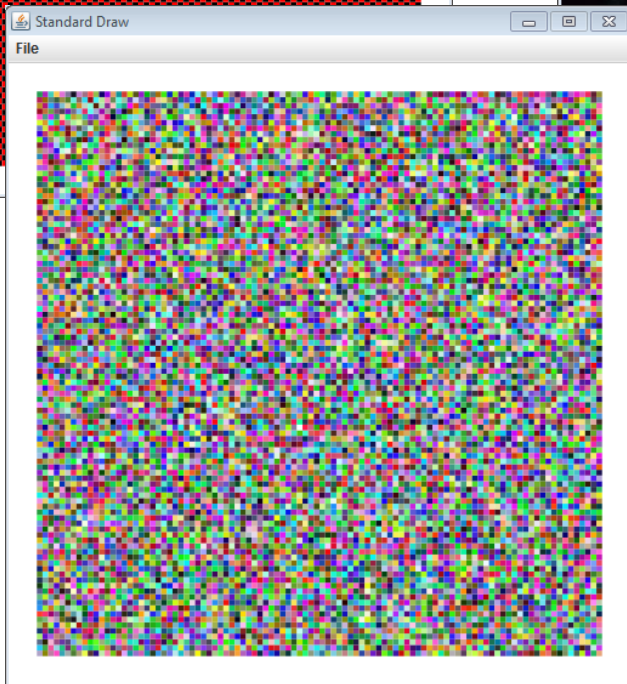
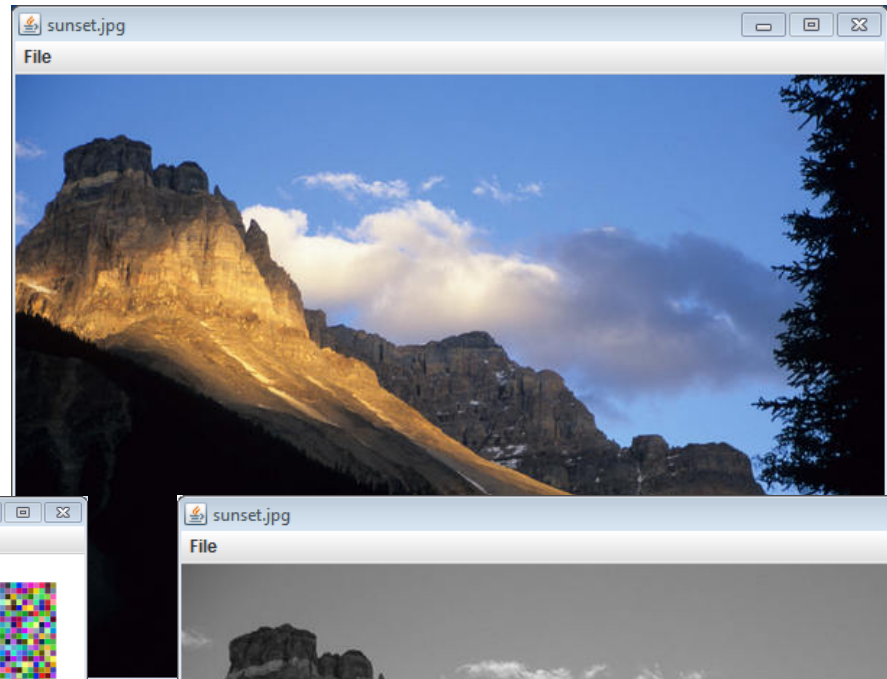
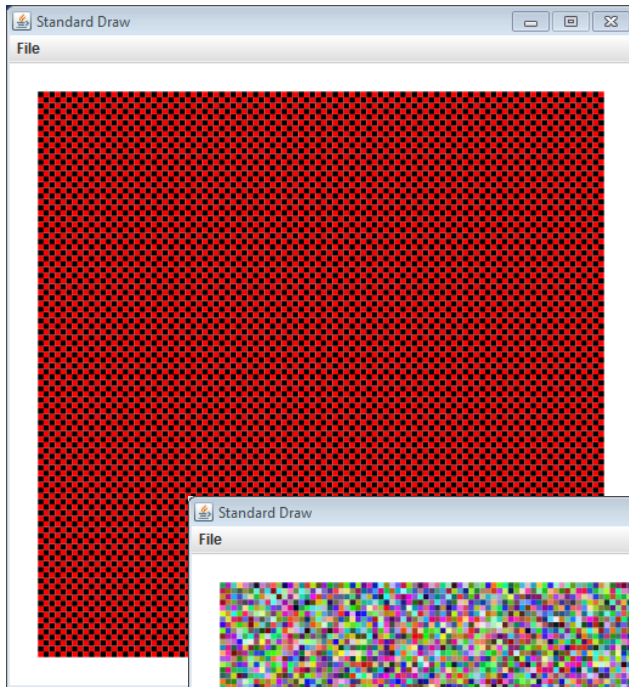


Using data types



Overview

- Using data types
 - What they are
 - Constructors and methods
 - Application Programming Interface (API)
 - Image processing
 - Text processing
- A slight diversion:
 - Incrementing and decrementing variables
- Methods
 - Parameter and return type puzzler

Data types

- Data type
 - "Set of values and operations on those values"
 - Set of values = what an object knows
 - Persistent state of an object
 - Operations on those values = what an object can do
 - Methods you can call on the object
- Contrast with primitive types:
 - Values map directly to machine representation
 - Operations directly map to machine instructions

Primitive type	Set of values	Operations
boolean	true, false	not, and, or, xor
int	-2^{31} to $2^{31}-1$	add, subtract, multiply, divide
double	any of 2^{64} possible reals	add, subtract, multiple, divide

Why custom data types?

- Data type
 - "Set of values and operations on those values"
 - Programs need to process other types of data:
 - e.g. colors, pictures, strings, complex numbers, vectors, matrices, polynomials, points, polygons, charged particles, celestial bodies, ...
- Big software projects
 - Non-trivial to manage dozens of programmers
 - Requires careful organization and structure
 - Divide project into small independent modules with clean and simple interfaces

Objects

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

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

Constructors and methods

- To construct a new object:
 - Use keyword `new` (to invoke 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)

Declare a variable
(object name)

Call a constructor
to create an object

```
String s;  
  
s = new String("Hello world!");  
  
System.out.println(s.substring(0, 5));
```

object name

call a method that operates
on the object's value

Image processing

- Color data type

- Color = sensation in eye from electromagnetic radiation

- Set of values:

- RGB representation, 256^3 possible values
 - Quantify amount of red, green, and blue, scale 0-255








R	G	B	Color
255	0	0	
0	255	0	
0	0	255	
255	255	255	
0	0	0	
255	0	255	
105	105	105	

Image processing

- Color data type

- Application Programming Interface (API)

- Public specification for what a data type does
 - All a program needs to know to use data type
 - Consists of: signature, return type, and comment for all public methods

```
public class java.awt.Color
```

	Color(int r, int g, int b)	
int	getRed()	<i>red intensity</i>
int	getGreen()	<i>green intensity</i>
int	getBlue()	<i>blue intensity</i>
Color	brighter()	<i>brighter version of this color</i>
Color	darker()	<i>darker version of this color</i>
String	toString()	<i>string representation of this color</i>
boolean	equals(Color c)	<i>is this color's value the same as c's?</i>

<http://docs.oracle.com/javase/7/docs/api/java/awt/Color.html>

Using Color data type

- **Goal:** Draw randomly colored checkerboard
 - User specifies grid size on command line
- **Approach:**
 - Modify our red and black checkerboard program

```
int N = Integer.parseInt(args[0]);
StdDraw.setXscale(0, N);
StdDraw.setYscale(0, N);

for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j++)
    {
        if ((i + j) % 2 != 0) StdDraw.setPenColor(StdDraw.BLACK);
        else StdDraw.setPenColor(StdDraw.RED);
        StdDraw.filledSquare(i + .5, j + .5, .5);
    }
}
```

Using Color data type

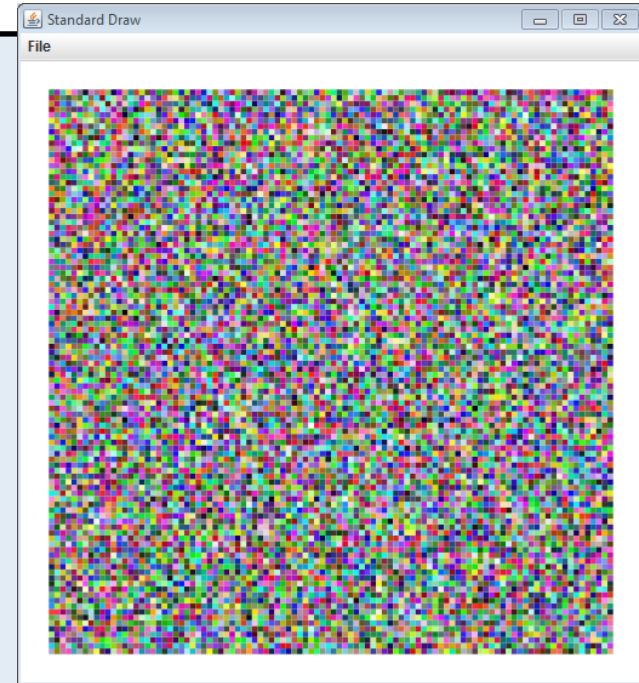
- **Goal: Draw randomly colored checkerboard**
 - User specifies grid size on command line

```
int N = Integer.parseInt(args[0]);
StdDraw.setXscale(0, N);
StdDraw.setYscale(0, N);

for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j++)
    {
        int r = (int) (Math.random() * 256);
        int g = (int) (Math.random() * 256);
        int b = (int) (Math.random() * 256);

        Color c = new Color(r, g, b);
        StdDraw.setPenColor(c);

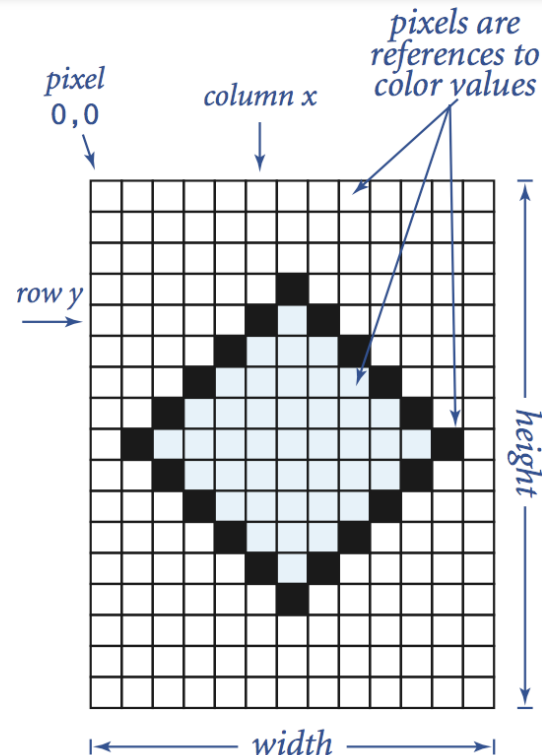
        StdDraw.filledSquare(i + .5, j + .5, .5);
    }
}
```



API for object representing an image

```
public class Picture
```

```
-----  
    Picture(String filename)    // create a picture from a file  
    Picture(int w, int h)      // create a blank w-by-h picture  
    int width()                // return the width of the picture  
    int height()               // return the height of the picture  
    Color get(int i, int j)    // return the color of pixel (i,j)  
    void set(int i, int j, Color c) // set the color of pixel (i,j) to c  
    void show()                // display the image in a window  
    void save(String filename) // save the image to a file
```



Using Color data type

- **Goal: Convert a color photo to greyscale**
 - Greyscale = RGB with the same values

R	G	B	Color
255	0	0	Red
0	255	0	Green
0	0	255	Blue
255	255	255	White
0	0	0	Black
255	0	255	Magenta
105	105	105	Grey



Using Color data type

- **Goal: Convert a color photo to greyscale**
 - How do we perceive the brightness of a color?
 - NTSC formula: $Y = 0.299r + 0.587g + 0.114b$

```
import java.awt.Color;

public class Luminance
{
    // return the monochrome luminance of given color
    public static double lum(Color color)
    {
        int r = color.getRed();
        int g = color.getGreen();
        int b = color.getBlue();

        return .299*r + .587*g + .114*b;
    }
}
```

Color compatibility

- Practical use of luminance:
 - Which font colors will be most readable with which background colors?
 - Good rule: difference in luminance ≥ 128






```
// are the two colors compatible?  
public static boolean compatible(Color a, Color b)  
{  
    return Math.abs(Lum(a) - Lum(b)) >= 128.0;  
}
```

Grayscale

- How to convert to grayscale?
 - Use luminance to determine R, G and B values

```
// return a gray version of this Color
public static Color toGray(Color color)
{
    int y = (int) (Math.round(lum(color))); // round to nearest int
    Color gray = new Color(y, y, y);
    return gray;
}
```

<i>red</i>	<i>green</i>	<i>blue</i>		
9	90	166	<i>this color</i>	
74	74	74	<i>grayscale version</i>	
0	0	0	<i>black</i>	

$$0.299 * 9 + 0.587 * 90 + 0.114 * 166 = 74.445$$

Grayscale filter program

- Goal: Convert a color photo to greyscale
- Our solution:
 - Use the data types: Picture, Color
 - Use static methods in the library: Luminance

```
Picture pic = new Picture(args[0]);
int width  = pic.width();
int height = pic.height();

// convert to grayscale
for (int x = 0; x < width; x++)
{
    for (int y = 0; y < height; y++)
    {
        Color color = pic.get(x, y);
        Color gray = Luminance.toGray(color);
        pic.set(x, y, gray);
    }
}
pic.show();
```


Text processing

- String data type
 - Basis for text processing
 - Set of values = sequence of Unicode characters

public class String (Java string data type)

String(String s)	<i>create a string with the same value as s</i>
int length()	<i>string length</i>
char charAt(int i)	<i>ith character</i>
String substring(int i, int j)	<i>ith through (j-1)st characters</i>
boolean contains(String sub)	<i>does string contain sub as a substring?</i>
boolean startsWith(String pre)	<i>does string start with pre?</i>
boolean endsWith(String post)	<i>does string end with post?</i>
int indexOf(String p)	<i>index of first occurrence of p</i>
int indexOf(String p, int i)	<i>index of first occurrence of p after i</i>
String concat(String t)	<i>this string with t appended</i>
int compareTo(String t)	<i>string comparison</i>
String replaceAll(String a, String b)	<i>result of changing as to bs</i>
String[] split(String delim)	<i>strings between occurrences of delim</i>
boolean equals(String t)	<i>is this string's value the same as t's?</i>

Typical string processing code

Is the string a palindrome?

```
public static boolean isPalindrome(String s)
{
    int N = s.length();
    for (int i = 0; i < N / 2; i++)
    {
        if (s.charAt(i) != s.charAt(N-1-i))
            return false;
    }
    return true;
}
```

*Extract file name and
extension from
command-line argument.*

```
String s = args[0];
int dot = s.indexOf(".");
String base = s.substring(0, dot);
String extension = s.substring(dot + 1, s.length());
```

Typical string processing code

Print all lines from standard input that contain a string specified on the command line.

```
String query = args[0];  
while (!StdIn.isEmpty())  
{  
    String s = StdIn.readLine();  
    if (s.contains(query))  
        System.out.println(s);  
}
```

Print all hyperlinks (to educational institutions) contained in the text received on standard input.

```
while (!StdIn.isEmpty())  
{  
    String s = StdIn.readString();  
    if (s.startsWith("http://") && s.endsWith("edu"))  
        System.out.println(s);  
}
```

Gene finding

- Genomics

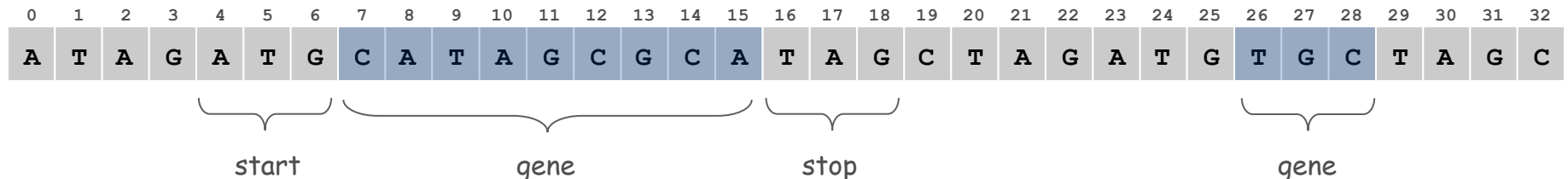
- Represent genome as string over alphabet: A C T G

- Gene

- Genome substring representing a functional unit

- Preceded by ATG [start codon]
 - Multiple of 3 nucleotides [codons other than start/stop]
 - Succeeded by TAG, TAA or TGA [stop codons]

–Goal: Find all genes



Gene finding: algorithm

- Algorithm:
 - Scan left-to-right through genome
 - If start codon, set beg to index i
 - If stop codon and substring multiple of 3
 - Output gene
 - Reset beg to -1

i	codon		beg	gene	<i>remaining portion of input string</i>
	<i>start</i>	<i>stop</i>			
0			-1		ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
1		TAG	-1		ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
4	ATG		4		ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
9		TAG	4		ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
16		TAG	4	CATAGCGCA	ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
20		TAG	-1		ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
23	ATG		23		ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
29		TAG	23	TGC	ATAGATGCATAGCGCATAGCTAGATGTGCTAGC

Gene finding: implementation

```
public class GeneFind
{
    public static void main(String[] args)
    {
        String start = args[0];
        String stop = args[1];
        String genome = StdIn.readAll();

        int beg = -1;
        for (int i = 0; i < genome.length() - 2; i++)
        {
            String codon = genome.substring(i, i+3);
            if (codon.equals(start)) beg = i;
            if ((codon.equals(stop)) && (beg != -1) && (beg + 3 < i))
            {
                String gene = genome.substring(beg+3, i);
                if (gene.length() % 3 == 0)
                {
                    System.out.println(gene);
                    beg = -1;
                }
            }
        }
    }
}
```

```
% more genomeTiny.txt
```

```
ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
```

```
% java GeneFind ATG TAG < genomeTiny.txt
```

```
CATAGCGCA
```

```
TGC
```


Increment and decrement

```
x = x + 1;
```

```
x += 1;
```

```
x++;
```

```
++x;
```




Each line increments
x by one.

```
x = x - 1;
```

```
x -= 1;
```

```
x--;
```

```
--x;
```



Each line decrements
x by one.

Pre and post increment/decrement

```
++x;  
--x;
```

prefix

increment/decrement
evaluates to value after the change

```
x++;  
x--;
```

postfix

increment/decrement
evaluates to value before the change

- If used on a line by itself, no difference
 - Use whichever one you fancy
 - Otherwise, you better know what you are doing!

```
int x = 0;  
int z = ++x;  
System.out.println("x=" + x +  
                    ", z=" + z);
```

x=1, z=1

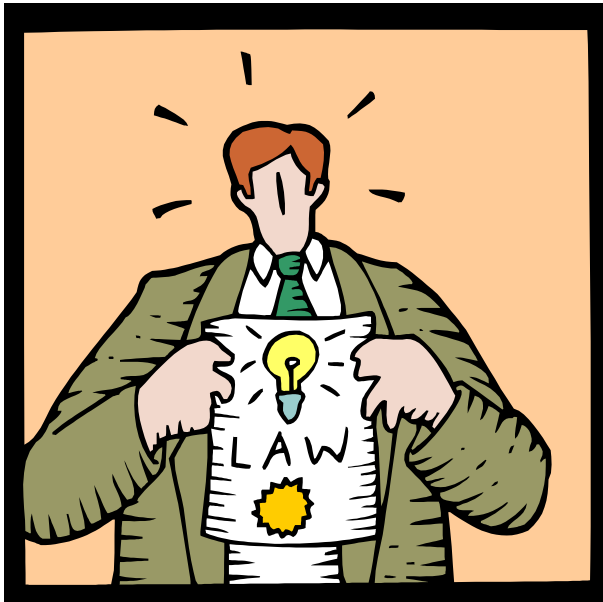
```
int x = 0;  
int z = x++;  
System.out.println("x=" + x +  
                    ", z=" + z);
```

x=1, z=0

Calling methods puzzler

```
int calcArea(int height, int width)
{
    return height * width;
}
```

Given the method above,
which of the methods calls on
the right are legal?



- 1) `int a = calcArea(7, 12);`
- 2) `short c = 7;`
`calcArea(c, 15);`
- 3) `int d = calcArea(57);`
- 4) `calcArea(2, 3);`
- 5) `long t = 42;`
`int f = calcArea(t, 17);`
- 6) `int g = calcArea();`
- 7) `calcArea();`
- 8) `byte h = calcArea(4, 20)`
- 9) `int j = calcArea(2, 3, 5);`
- 10) `int k = calcArea(2.0, 2.0);`

```
int calcArea(int height, int width)
{
    return height * width;
}
```

Wrong number of arguments to method. We must pass exactly two parameters!

```
1) int a = calcArea(7, 12);
2) short c = 7;
   calcArea(c, 15);
3) int d = calcArea(57);
4) calcArea(2, 3);
5) long t = 42;
   int f = calcArea(t, 17);
6) int g = calcArea();
7) calcArea();
8) byte h = calcArea(4, 20);
9) int j = calcArea(2, 3, 5);
10) int k = calcArea(2.0, 2.0);
```

```
int calcArea(int height, int width)
{
    return height * width;
}
```

Parameter type problem.
A long won't fit into an int
parameters without spilling.

Return type problem.
Method returns an int
which won't fit into a byte
without spilling.

Parameter type problem.
The double's won't demote
to lowly int parameters.

```
1) int a = calcArea(7, 12);
2) short c = 7;
   calcArea(c, 15);
3) int d = calcArea(57);
4) calcArea(2, 3);
5) long t = 42;
   int f = calcArea(t, 17);
6) int g = calcArea();
7) calcArea();
8) byte h = calcArea(4, 20)
9) int j = calcArea(2, 3, 5);
10) int k = calcArea(2.0, 2.0);
```

```
int calcArea(int height, int width)
{
    return height * width;
}
```

Lovely. Just how we'd expect somebody to do it!

First parameter is a short but it can fit in an int parameter since it is a bigger data type.

Sort of weird but it will compile. We get an int result back, but we just ignore it.

1) ~~int~~ a = calcArea(7, 12);

2) ~~short~~ c = 7;
calcArea(c, 15);

3) ~~int~~ d = calcArea(57);

4) calcArea(2, 3);

5) ~~long~~ t = 42;
~~int~~ f = calcArea(t, 17);

6) ~~int~~ g = calcArea();

7) calcArea();

8) ~~byte~~ h = calcArea(4, 20);

9) ~~int~~ j = calcArea(2, 3, 5);

10) ~~int~~ k = calcArea(2.0, 2.0);

```
double calcArea(double height,  
                double width)  
{  
    return height * width;  
}
```

Which are legal if instead the method took two double's and returned a double?



- 1) **int** a = calcArea(7, 12);
- 2) **short** c = 7;
 calcArea(c, 15);
- 3) **double** d = calcArea(7.0, 2);
- 4) **double** e = calcArea(7, 2.0);
- 5) **double** f = calcArea(7.2, 2.0);
- 6) **int** g = calcArea(7.2, 2.0);
- 7) **float** h = 1.99f;
 double i = calcArea(h, h);
- 8) **double** j = calcArea("7.0",
 "12.0");
- 9) **String** k = "" + calcArea(1, 2);
- 10) **double** m = calcArea(-1.0, -9.0);

```
double calcArea(double height,  
                double width)  
{  
    return height * width;  
}
```

Parameters 7 and 12
promote to double,
but return value can't
demote to an int.

Parameters are fine,
but return value can't
demote to an int.

```
1) int a = calcArea(7, 12);  
2) short c = 7;  
   calcArea(c, 15);  
3) double d = calcArea(7.0, 2);  
4) double e = calcArea(7, 2.0);  
5) double f = calcArea(7.2, 2.0);  
6) int g = calcArea(7.2, 2.0);  
7) float h = 1.99f;  
   double i = calcArea(h, h);  
8) double j = calcArea("7.0",  
                       "12.0");  
9) String k = "" + calcArea(1, 2);  
10) double m = calcArea(-1.0, -9.0);
```

```
double calcArea(double height,  
                double width)  
{  
    return height * width;  
}
```

Parameters are of type
String and won't convert
to double without a call to
Double.parseDouble()

```
1) int a = calcArea(7, 12);  
2) short c = 7;  
   calcArea(c, 15);  
3) double d = calcArea(7.0, 2);  
4) double e = calcArea(7, 2.0);  
5) double f = calcArea(7.2, 2.0);  
6) int g = calcArea(7.2, 2.0);  
7) float h = 1.99f;  
   double i = calcArea(h, h);  
8) double j = calcArea("7.0",  
                        "12.0");  
9) String k = "" + calcArea(1, 2);  
10) double m = calcArea(-1.0, -9.0);
```

```
double calcArea(double height,  
                double width)  
{  
    return height * width;  
}
```

Types such as short, int, and float will all type promote to double if needed.

The double return result can be appended to a String. using + (but we must have the blank string "" first).

```
1) int a = calcArea(7, 12);  
2) short c = 7;  
   calcArea(c, 15);  
3) double d = calcArea(7.0, 2);  
4) double e = calcArea(7, 2.0);  
5) double f = calcArea(7.2, 2.0);  
6) int g = calcArea(7.2, 2.0);  
7) float h = 1.99f;  
   double i = calcArea(h, h);  
8) double j = calcArea("7.0",  
                        "12.0");  
9) String k = "" + calcArea(1, 2);  
10) double m = calcArea(-1.0, -9.0);
```


Summary

- Data type
 - "Set of values and operations on those values"
 - Declaring/creating variables of a custom data type
 - Use the new operator
 - Allows us to build bigger, more complex software
- Using data types
 - Knowing the API, we can create useful programs:
 - e.g. converting images to grayscale, finding genes
- Incrementing/decrementing variables
 - More than one way to do it
 - Be careful if you rely on prefix/postfix semantics!