

INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

9. Abstract Data Types



<http://introcs.cs.rutgers.edu>



9. Abstract Data Types

- Overview
- Color
- Image processing
- String processing

Abstract data types

Explain the meaning of *data type*.

LO 9.1a

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

Primitive types

- *values* immediately map to machine representations
- *operations* immediately map to machine instructions.

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

- Colors, pictures, strings,
- Complex numbers, vectors, matrices,
- ...

An **abstract data type** is a data type whose representation is hidden from the client.

Built-in data types

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

type	set of values	examples of values	examples of operations
char	characters	'A' '@'	compare
String	sequences of characters	"Hello World" "CS is fun"	concatenate
int	integers	17 12345	add, subtract, multiply, divide
double	floating-point numbers	3.1415 6.022e23	add, subtract, multiply, divide
boolean	truth values	true false	and, or, not

Java's built-in data types

Object-oriented programming (OOP)

Object-oriented programming (OOP).

- Create your own data types.
- Use them in your programs (manipulate *objects*).

An **object** holds a data type value.
Variable names refer to objects.



Examples (stay tuned for details)

<i>data type</i>	<i>set of values</i>	<i>examples of operations</i>
Color	three 8-bit integers	get red component, brighten
Picture	2D array of colors	get/set color of pixel (i, j)
String	sequence of characters	length, substring, compare



Best practice: Use *abstract* data types (representation is *hidden from the client*).

Impact: Clients can use ADTs without knowing implementation details.

- This lecture: how to write client programs for several useful ADTs
- Next lecture: how to implement your own ADTs

Sound

We have *already* been using ADTs!

Crash course in sound

Sound is the perception of the vibration of molecules in our eardrums.

A musical tone is a steady periodic sound.

A pure tone is a sinusoidal waveform.

Western musical scale

- Concert A is 440 Hz.
- 12 notes, logarithmic scale.



pitch	i
A	0
A# / Bb	1
B	2
C	3
C# / Db	4
D	5
D# / Eb	6
E	7
F	8
F# / Gb	9
G	10
G# / Ab	11
A	12

Digital audio

To represent a wave, *sample* at regular intervals and save the values in an array.

← same as when plotting a function (previous lecture)

	samples/sec	samples	sampled waveform
1/40 second of concert A	5,512	137	
	11,025	275	
	22,050	551	
CD standard	44,100	1102	

Bottom line. You can *write programs* to manipulate sound (arrays of double values).

Sound ADT

Values: Array of doubles.

Operations: specified in **API**.

public class StdAudio	
void play(double[] a)	<i>play the given sound wave</i>
void save(String file, double[] a)	<i>save to a .wav file</i>
double[] read(String file)	<i>read from a .wav file</i>

Representation: Hidden from user (.wav and other formats needed by devices).

Strings

Write code that
manipulates/processes Strings by
calling String methods.

LO 9.1e

stay tuned for more complete API later in this lecture

We have *already* been using ADTs!

A **String** is a sequence of Unicode characters.

Java's **String ADT** allows us to write Java programs that manipulate strings. The exact representation is hidden (it could change and our programs would still work).

public class String	
String(String s)	<i>create a string with the same value</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?</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 1st occurrence of p</i>
int indexOf(String p, int i)	<i>index of 1st 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>

Operations (API)

Using a data type: constructors and methods

To **use** a data type, you need to know:

- Its name (capitalized, in Java).
- How to *construct* new objects.
- How to *apply operations* to a given object.

To **construct a new object**

- Use the keyword **new** to invoke a *constructor*.
- Use **data type name** to specify type of object.

To **apply an operation (invoke a method)**

- Use **object name** to specify which object.
- Use the **dot operator** to indicate that an operation is to be applied.
- Use a **method name** to specify which operation.



new Building()

```
String s;  
s = new String ("Hello, World");  
StdOut.println( s.substring(0, 5) );
```

Q. What is a data type?

A. A set of values and a set of operations on those values.

Q. What is an abstract data type?

Pop quiz on ADTs

Q. What is a data type?

A. A set of values and a set of operations on those values.

Q. What is an abstract data type?

A. A data type whose representation is hidden from the client.

Primitive data type versus object reference

Distinguish between primitive data types and object references.

LO 9.1c

Q. What is a primitive data type?

A. A set primitive values (int, double, char, ..) that are not abstract data types.

Q. What is an object reference?

A. A reference to the object but not the internal details of the object.

Java provides object types for all primitive types



<i>primitive type</i>	<i>wrapper type</i>
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Immutability of Strings

Explain the immutability of
Strings.

LO 9.1f

Q. What is an immutable String?

A. An object whose individual characters cannot be changed

Q. What is a mutable object?

A. An object whose internal representation can be altered.

Immutable objects have
locked internal structure →

<i>immutable</i>	<i>mutable</i>
String	Turtle
Charge	Picture
Color	Histogram
Complex	StockAccount
Vector	Counter
	Java arrays

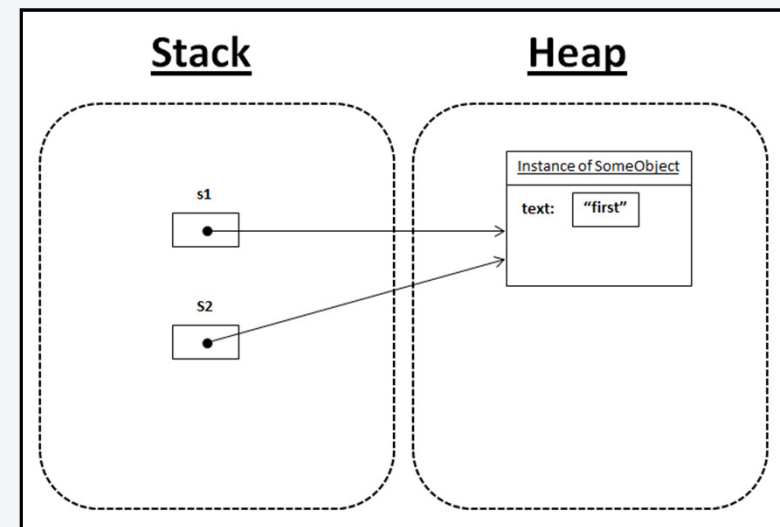
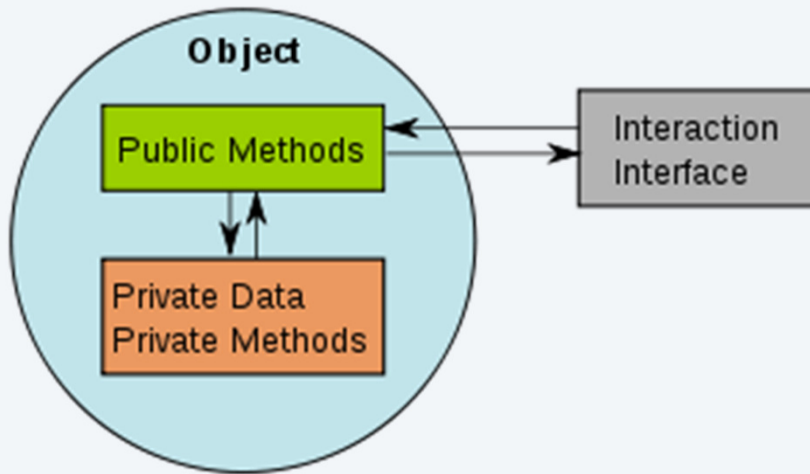
↗
Mutable objects have
instance fields that can
be changed

Objects and References

Distinguish
between *object* and
reference.

LO 9.1b

An object is an instance of a class that is created by using the “new” constructor.



The reference to an object is location information to the actual object stored in stack.



9. Abstract Data Types

- Overview
- Color
- Image processing
- String processing









Color ADT

Color is a sensation in the eye from electromagnetic radiation.



An **ADT** allows us to write Java programs that manipulate color.

Values

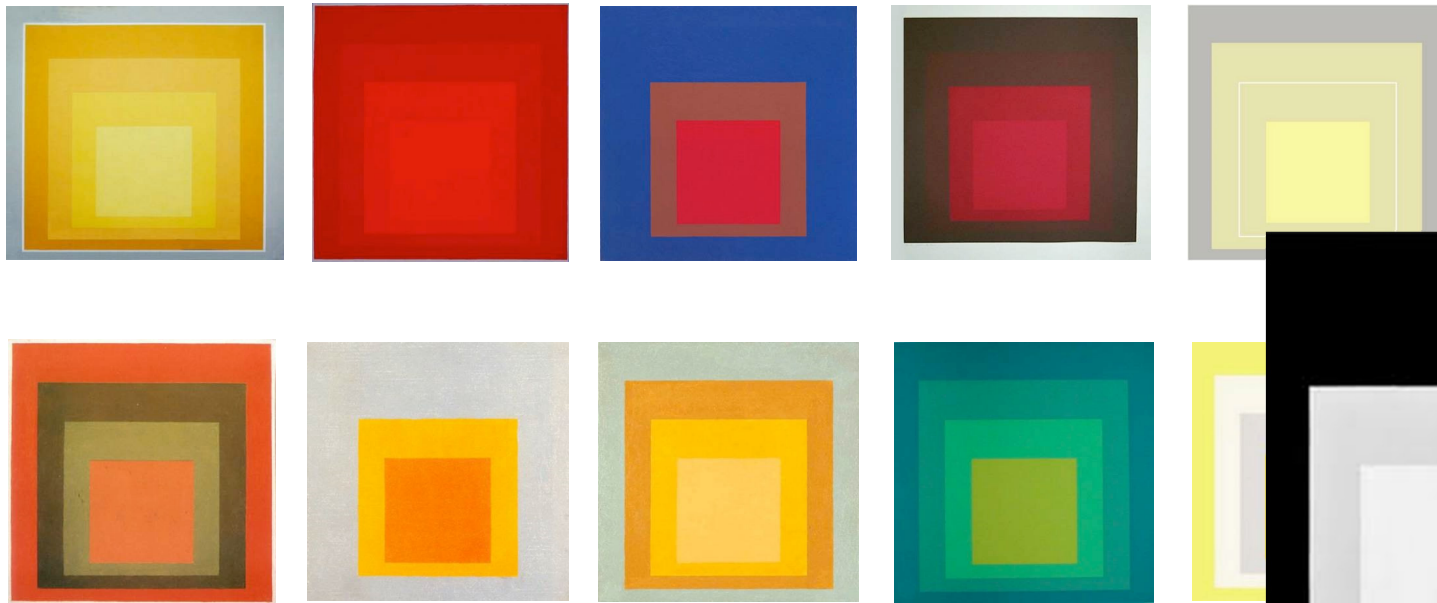
		examples							
R (8 bits)	red intensity	255	0	0	0	255	0	119	105
G (8 bits)	green intensity	0	255	0	0	255	64	33	105
B (8 bits)	blue intensity	0	0	255	0	255	128	27	105
color									

API (operations)

public class 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 the same as c's ?</i>

Albers squares

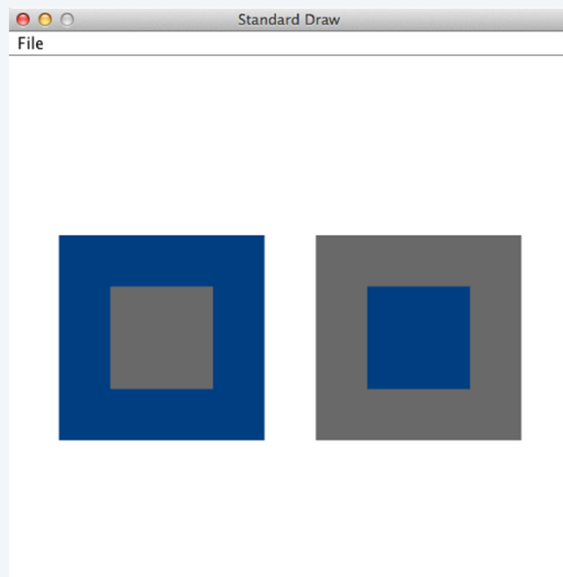
Josef Albers. A 20th century artist who revolutionized the way people think about color.



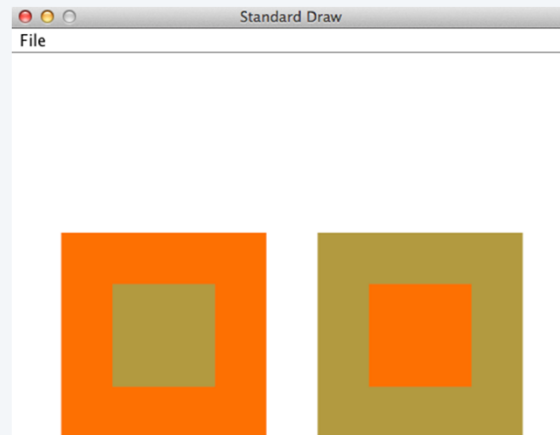
Josef Albers 1888–1976

Color client example: Albers squares

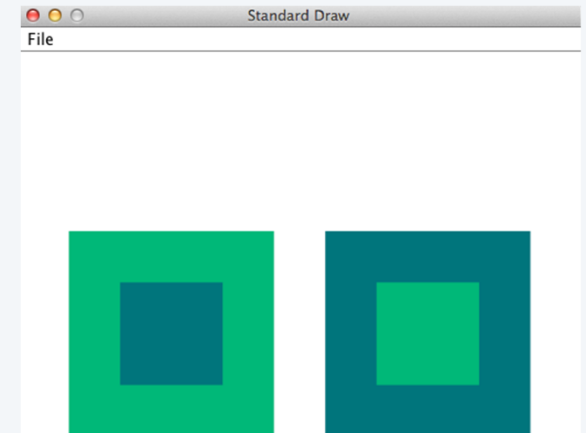
Goal. Write a Java program to generate Albers squares.



```
% java AlbersSquares 0 64 128 105 105 105
```



```
% java AlbersSquares 251 112 34 177 153 71
```

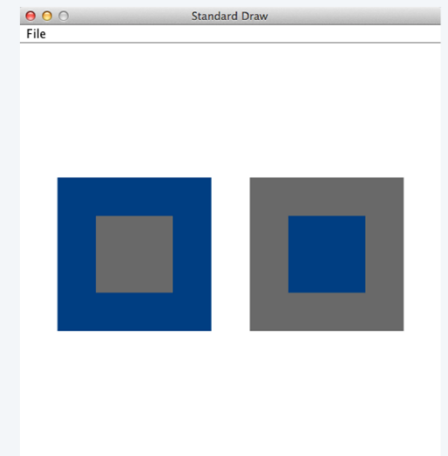
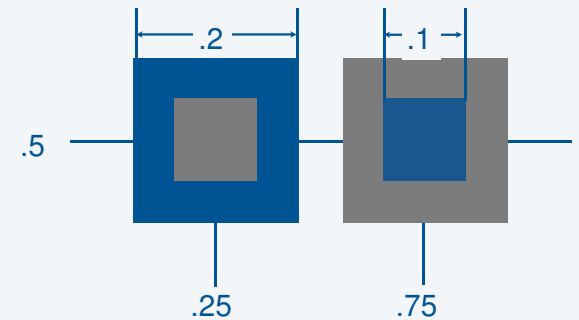


```
% java AlbersSquares 28 183 122 15 117 123
```


Color client example: Albers squares

```
public class AlbersSquares {  
  
    public static void main(String[] args) {  
        int r1 = Integer.parseInt(args[0]);  
        int g1 = Integer.parseInt(args[1]);  
        int b1 = Integer.parseInt(args[2]);  
        Color c1 = new Color(r1, g1, b1);  
        int r2 = Integer.parseInt(args[3]);  
        int g2 = Integer.parseInt(args[4]);  
        int b2 = Integer.parseInt(args[5]);  
        Color c2 = new Color(r2, g2, b2);  
        StdDraw.setPenColor(c1);  
        StdDraw.filledSquare(.25, .5, .2);  
        StdDraw.setPenColor(c2);  
        StdDraw.filledSquare(.25, .5, .1);  
        StdDraw.setPenColor(c2);  
        StdDraw.filledSquare(.75, .5, .2);  
        StdDraw.setPenColor(c1);  
        StdDraw.filledSquare(.75, .5, .1);  
    }  
}
```

```
% java AlbersSquares 0 64 128 105 105 105
```











Computing with color: monochrome luminance

Def. The *monochrome luminance* of a color quantifies its **effective brightness**.

NTSC standard formula for luminance: $0.299r + 0.587g + 0.114b$.

```
import java.awt.Color;
public class Luminance
{
    public static double lum(Color c)
    {
        int r = c.getRed();
        int g = c.getGreen();
        int b = c.getBlue();
        return .299*r + .587*g + .114*b;
    }
    public static void main(String[] args)
    {
        int r = Integer.parseInt(args[0]);
        int g = Integer.parseInt(args[1]);
        int b = Integer.parseInt(args[2]);
        Color c = new Color(r, g, b);
        StdOut.println(Math.round(lum(c)));
    }
}
```

```
% java Luminance 0 64 128
52
```

	examples							
red intensity	255	0	0	0	255	0	119	105
green intensity	0	255	0	0	255	64	33	105
blue intensity	0	0	255	0	255	128	27	105
color								
luminance	76	150	29	0	255	52	58	105

Applications (next)

- Choose colors for displayed text.
- Convert colors to grayscale.

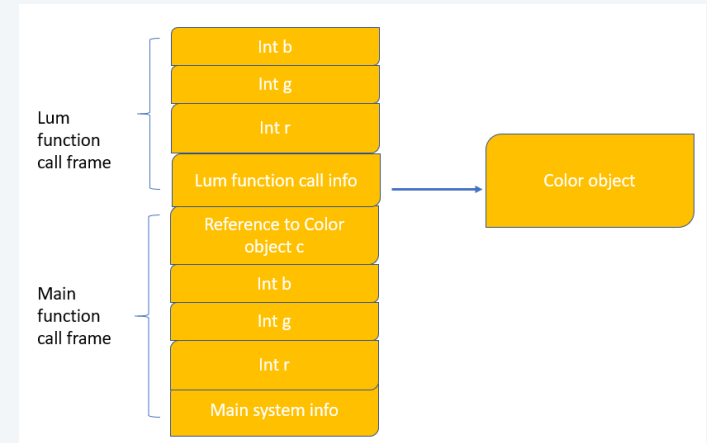
Execution and call stack

LO 9.1h

Given below is a client program that utilizes the Color API (slide 13).
Illustrate the call stack and memory allocation for the execution of the client program.

```
public static void main(String[] args)
{
    int r = Integer.parseInt(args[0]);
    int g = Integer.parseInt(args[1]);
    int b = Integer.parseInt(args[2]);
    Color c = new Color(r, g, b);
    StdOut.println(Math.round(lum(c)));
}
```

Given a class API, write a client program that creates objects of that class and call methods defined for that class and illustrate the call stack and memory allocation for the execution of this client program.



Computing with color: compatibility

Q. Which font colors will be most readable with which background colors on a display?

Rule of thumb. Absolute value of difference in luminosity should be > 128 .

```
public static boolean compatible(Color a, Color b)
{
    return Math.abs(lum(a) - lum(b)) > 128.0;
}
```

		76	0	255	52
	76	255	76	179	24
	0	76		255	52
	255	179	255	255	203
	52	24	52	203	

Computing with color: grayscale

Goal. Convert colors to grayscale values.

Fact. When all three R, G, and B values are the same, resulting color is on grayscale from 0 (black) to 255 (white).

















Q. What value for a given color?

A. Its **luminance**!

```
public static Color toGray(Color c)
{
    int y = (int) Math.round(lum(c));
    Color gray = new Color(y, y, y);
    return gray;
}
```

↑
method for Luminance library



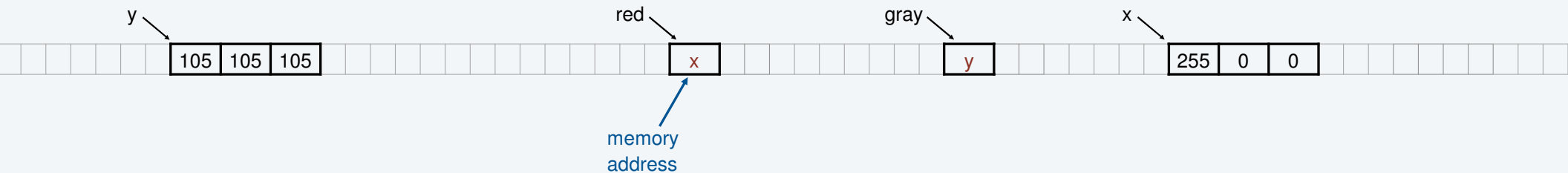
	examples							
red intensity	255	0	0	0	255	0	119	105
green intensity	0	255	0	0	255	64	33	105
blue intensity	0	0	255	0	255	128	27	105
color								
luminance	76	150	29	0	255	52	58	105
grayscale								

OOP context for color

Q. How does Java represent color? Three int values? Packed into one int value?

A. *We don't know.* The representation is hidden. It is an *abstract* data type.

Possible memory representation of `red = new Color(255, 0, 0)`
and `gray = new Color(105, 105, 105);`



An object reference is analogous to a variable name.

- It is not the value but it refers to the value.
- We can manipulate the value in the object it refers to.
- We can pass it to (or return it from) a method.

We also use object references to *invoke* methods (with the `.` operator)

References and abstraction

René Magritte. This is not a pipe.



← It is a picture of a painting of a pipe.

Java. These are not colors.

```
public static Color toGray(Color c)
{
    int y = (int) Math.round(lum(c));
    Color gray = new Color(y, y, y);
    return gray;
}
```

Object-oriented programming. A natural vehicle for studying abstract models of the real world.

"This is not a pipe."



Yes it is! He's referring to the physical object he's holding. Joke would be better if he were holding a *picture* of a pipe.

This is not a pipe.



Surrealist computer scientist:
Neither is this.

% java RandomSeq 10000 | java Average



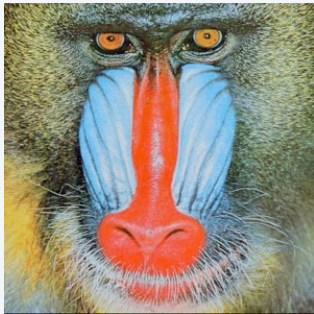
9. Abstract Data Types

- Overview
- Color
- Image processing
- String processing

Picture ADT

A **Picture** is a 2D array of pixels.

defined in terms of its ADT values (typical)

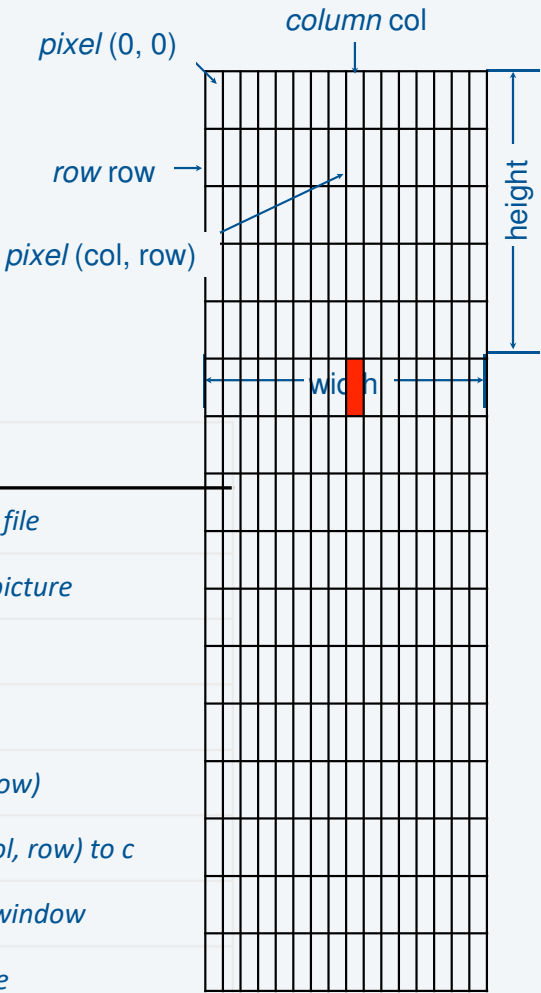


An **ADT** allows us to write Java programs that manipulate pictures.

API (operations)

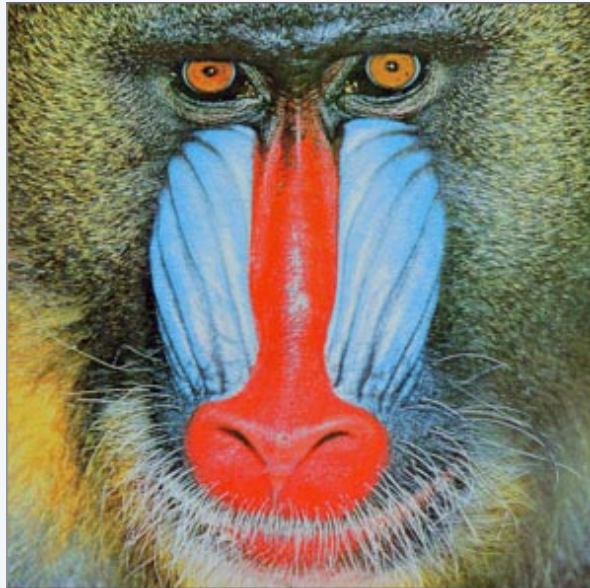
public class Picture		
Picture(String filename)		<i>create a picture from a file</i>
Picture(int w, int h)		<i>create a blank w-by-h picture</i>
int width()		<i>width of the picture</i>
int height()		<i>height of the picture</i>
Color get(int col, int row)		<i>the color of pixel (col, row)</i>
void set(int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

Values (2D arrays of Colors)

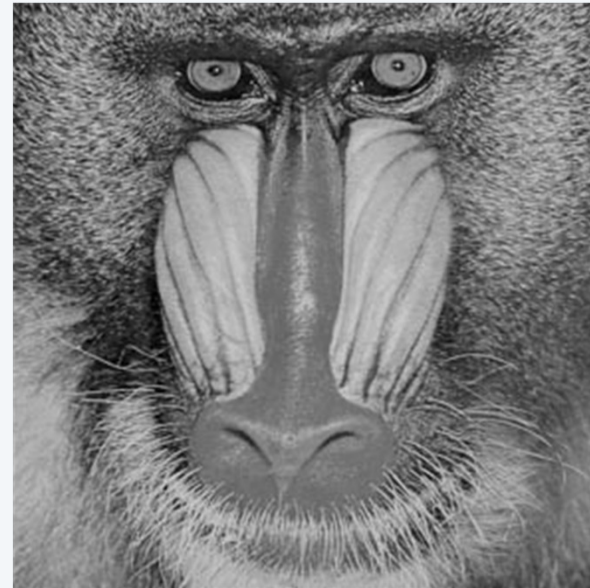


Picture client example: Grayscale filter

Goal. Write a Java program to convert an image to grayscale.



Source: [mandrill.jpg](#)



```
% java Grayscale mandrill.jpg
```

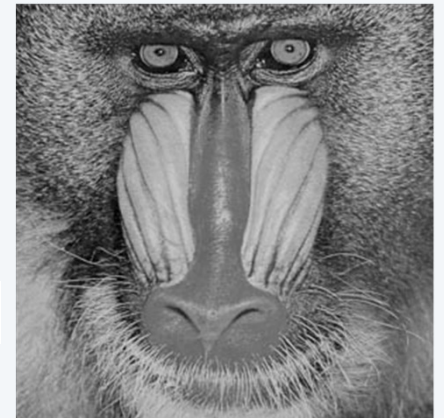
Picture client example: Grayscale filter

```
import java.awt.Color;
public class Grayscale
{
    public static void main(String[] args)
    {
        Picture pic = new Picture(args[0]);
        for (int col = 0; col < pic.width(); col++)
            for (int row = 0; row < pic.height(); row++)
            {
                Color color = pic.get(col, row);
                Color gray = Luminance.toGray(color);
                pic.set(col, row, gray);
            }
        pic.show();
    }
}
```

← create a new picture

← fill in each pixel

```
% java Grayscale mandrill.jpg
```



Pop quiz 1a on image processing

Q. What is the effect of the following code (easy question)?

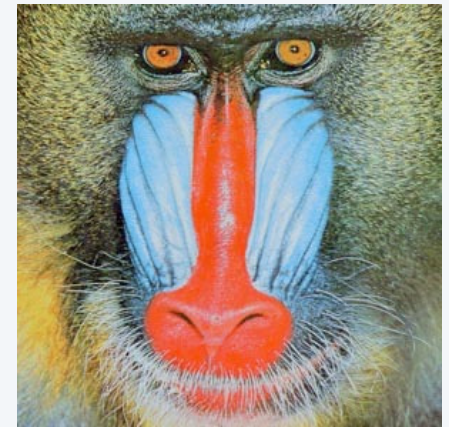
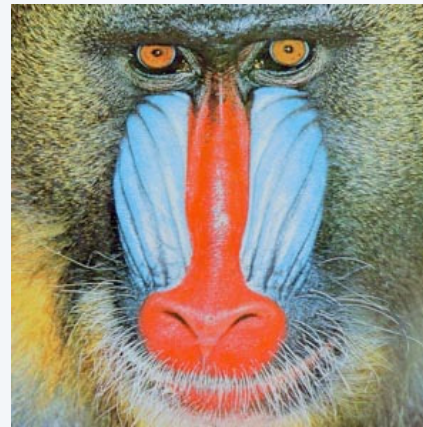
```
Picture pic = new Picture(args[0]);  
for (int col = 0; col < pic.width(); col++)  
    for (int row = 0; row < pic.height(); row++)  
        pic.set(col, row, pic.get(col, row));  
pic.show();
```

Pop quiz 1a on image processing

Q. What is the effect of the following code (easy question)?

```
Picture pic = new Picture(args[0]);  
for (int col = 0; col < pic.width(); col++)  
    for (int row = 0; row < pic.height(); row++)  
        pic.set(col, row, pic.get(col, row));  
pic.show();
```

A. None. Just shows the picture.



Pop quiz 1b on image processing

Q. What is the effect of the following code (not-so-easy question)?

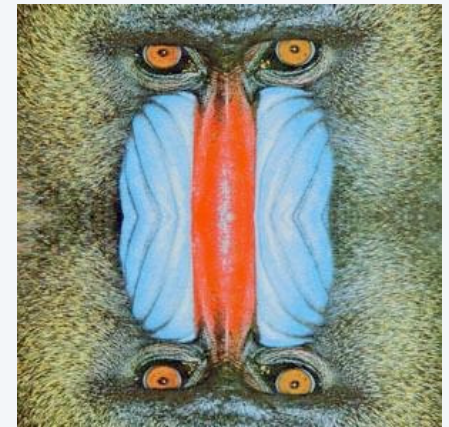
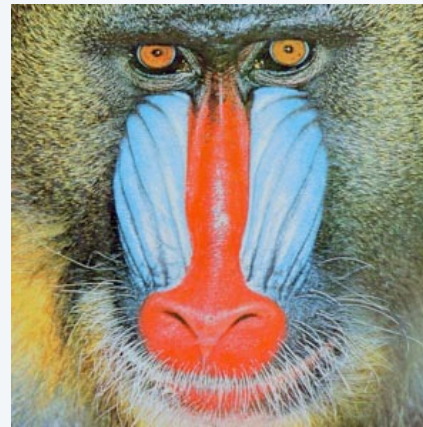
```
Picture pic = new Picture(args[0]);
for (int col = 0; col < pic.width(); col++)
    for (int row = 0; row < pic.height(); row++)
        pic.set(col, pic.height()-row-1, pic.get(col, row));
pic.show();
```

Pop quiz 1b on image processing

Q. What is the effect of the following code (not-so-easy question)?

```
Picture pic = new Picture(args[0]);  
for (int col = 0; col < pic.width(); col++)  
    for (int row = 0; row < pic.height(); row++)  
        pic.set(col, pic.height()-row-1, pic.get(col, row));  
pic.show();
```

A. Tries to turn image upside down, but fails.
An instructive bug!.



Pop quiz 1c on image processing

Q. What is the effect of the following code?

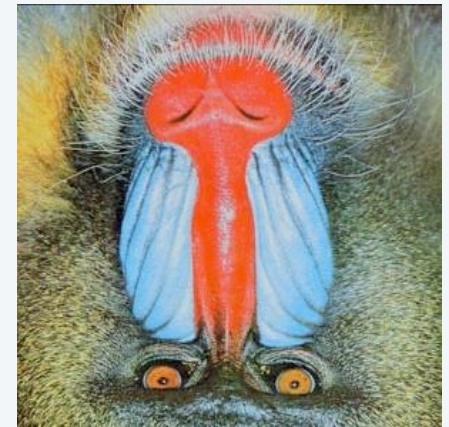
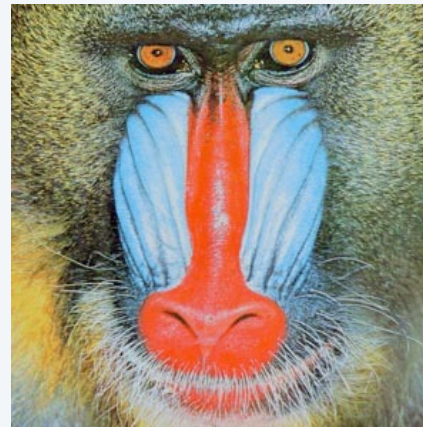
```
Picture source = new Picture(args[0]);
int width  = source.width();
int height = source.height();
Picture target = new Picture(width, height);
for (int col = 0; col < width; col++)
    for (int row = 0; row < height; row++)
        target.set(col, height-row-1, source.get(col, row));
target.show();
```

Pop quiz 1c on image processing

Q. What is the effect of the following code?

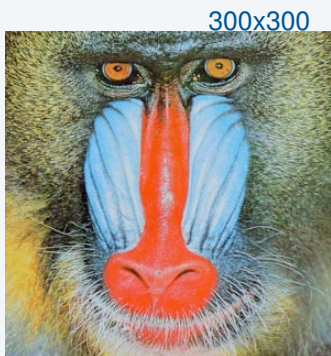
```
Picture source = new Picture(args[0]);  
int width = source.width();  
int height = source.height();  
Picture target = new Picture(width, height);  
for (int col = 0; col < width; col++)  
    for (int row = 0; row < height; row++)  
        target.set(col, height-row-1, source.get(col, row));  
target.show();
```

A. Makes an upside down copy of the image.

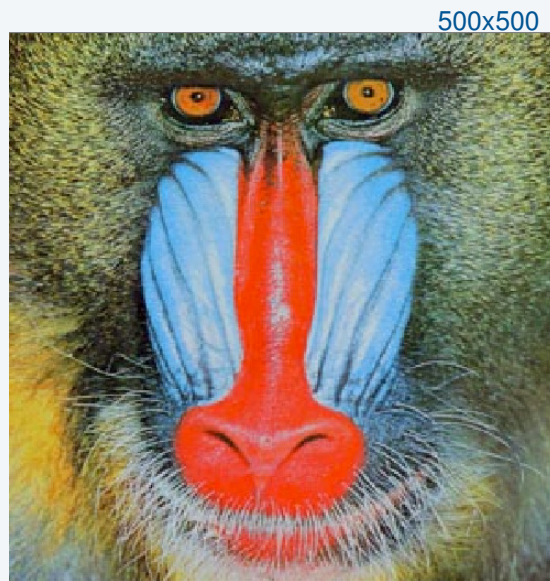


Picture client example: Scaling filter

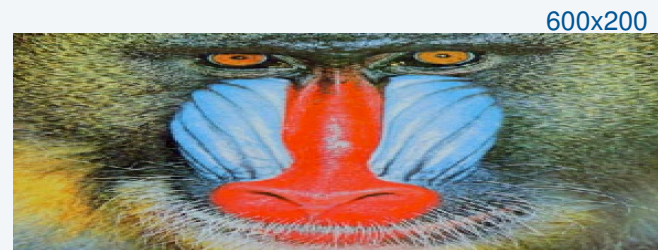
Goal. Write a Java program to scale an image (arbitrarily and independently on x and y).



Source: mandrill.jpg



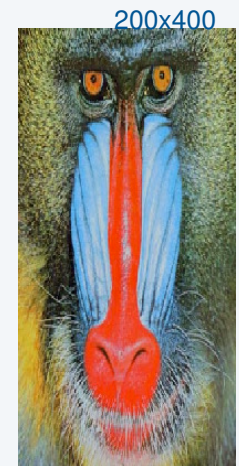
```
% java Scale mandrill.jpg 500 500
```



```
% java Scale mandrill.jpg 600 200
```



```
% java Scale mandrill.jpg 99 99
```



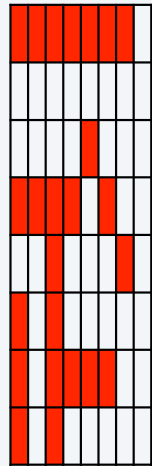
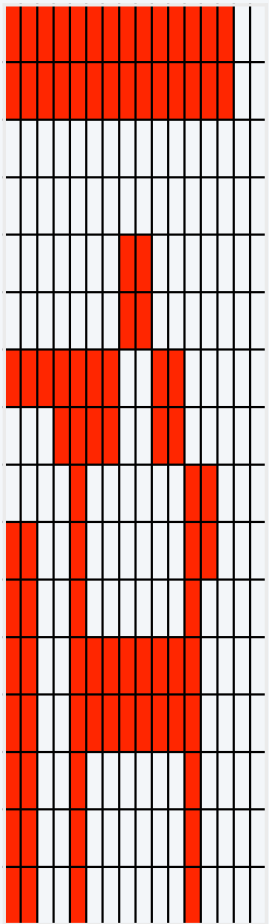
```
% java Scale mandrill.jpg 200 400
```

Picture client example: Scaling filter

Goal. Write a Java program to scale an image (arbitrarily and independently on x and y).

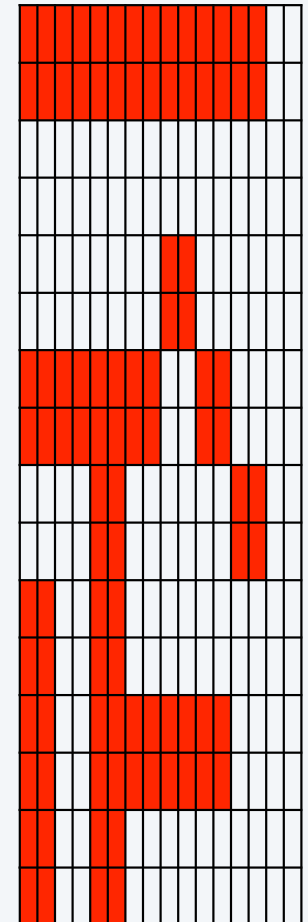
Ex. Downscaling by halving.

Shrink in half by deleting alternate rows and columns.



Ex. Upscaling by doubling.

Double in size by replacing each pixel with four copies.



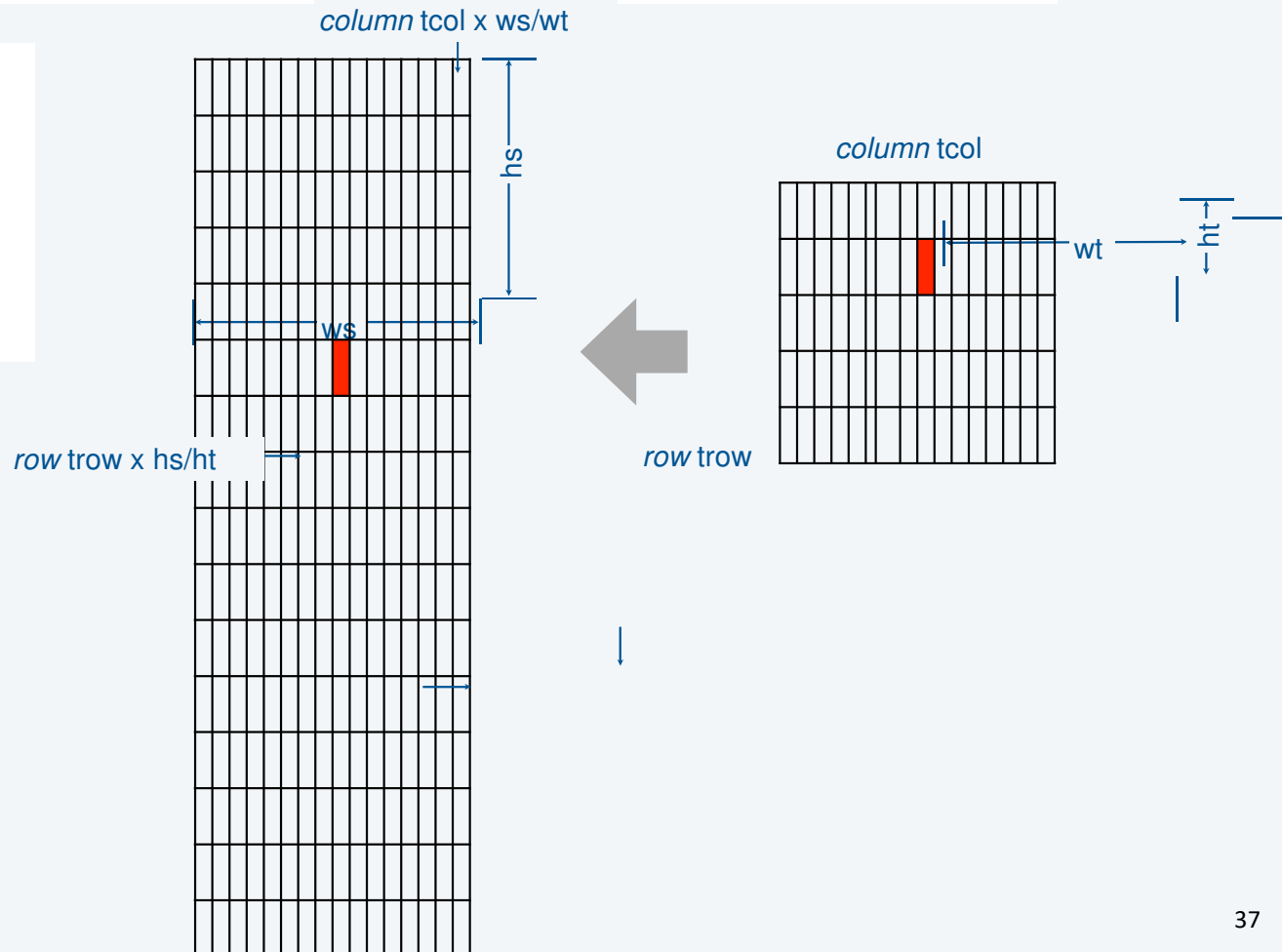
Picture client example: Scaling filter

Goal. Write a Java program to scale an image (arbitrarily and independently on x and y).

A uniform strategy to scale from ws -by- hs to wt -by- ht .

- Scale column index by ws/wt .
- Scale row index by hs/ht .

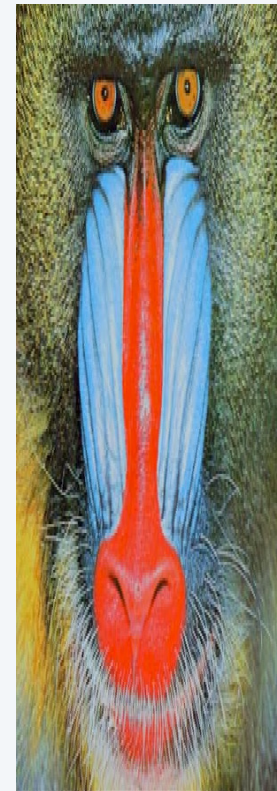
Approach. Arrange computation to compute exactly one value for each *target* pixel.



Picture client example: Scaling filter

```
import java.awt.Color;
public class Scale
{
    public static void main(String[] args)
    {
        String filename = args[0];
        int w = Integer.parseInt(args[1]);
        int h = Integer.parseInt(args[2]);
        Picture source = new Picture(filename);
        Picture target = new Picture(w, h);
        for (int tcol = 0; tcol < w; tcol++)
            for (int trow = 0; trow < h; trow++)
            {
                int scol = tcol * source.width() / w;
                int srow = trow * source.height() / h;
                Color color = source.get(scol, srow);
                target.set(tcol, trow, color);
            }
        target.show();
    }
}
```

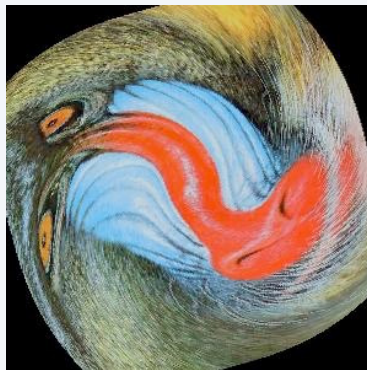
```
% java Scale mandrill.jpg 300 900
```



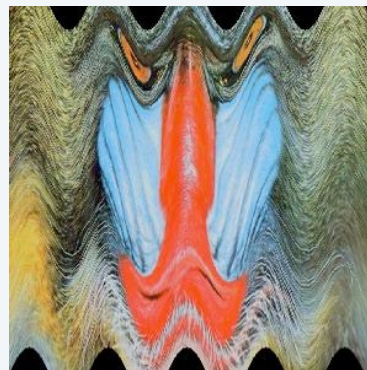
More image-processing effects



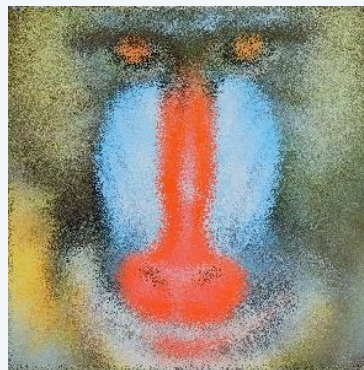
RGB color separation



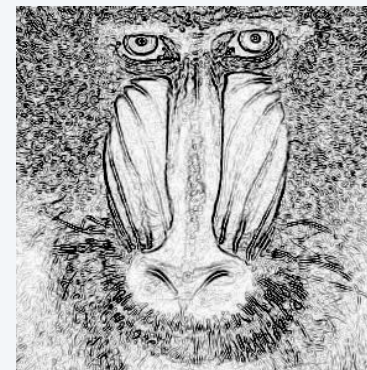
swirl filter



wave filter



glass filter



Sobel edge detection



9. Abstract Data Types

- Overview
- Color
- Image processing
- String processing

String ADT

Write code that manipulates/processes Strings by calling String methods.

LO 9.1e

A **String** is a sequence of Unicode characters.

← defined in terms of its ADT values (typical)

Java's **ADT** allows us to write Java programs that manipulate strings.

Operations (API)

public class String	
String(String s)	<i>create a string with the same value</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?</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(Object t)	<i>is this string's value the same as t's ?</i>

Concatenation of String

Explain the results of concatenating Strings with primitive data types using the + operator

LO 9.1g

A **String** concatenation is a common operation.

The operator + is overloaded for String concatenation.

```
String s1 = "Hello";
```

```
String s2 = "There";
```

```
String s3 = s1 + " " + s2
```

← Results in s3 = "Hello There"

Note. s3 is a new String object. The Strings s1, and s2 still exists.

Programming with strings: typical examples

Write code that manipulates/processes
Strings by calling String methods.

LO 9.1e

```
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;
}
```

Is the string a palindrome?

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

Find lines containing a specified string in StdIn

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

Search for *.edu hyperlinks in the text file on StdIn

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

Find occurrences of a specified string in StdIn

String client example: gene finding

Pre-genomics era. Sequence a human genome.

Post-genomics era. Analyze the data and understand structure.

Genomics. Represent genome as a string over A C T G alphabet.

Gene. A substring of genome that represents a functional unit.

- Made of *codons* (three A C T G *nucleotides*).
- Preceded by ATG (*start codon*).
- Succeeded by TAG, TAA, or TGA (*stop codon*).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
A	T	A	G	A	T	G	C	A	T	A	G	C	G	C	A	T	A	G	C	T	A	G	A	T	G	T	G	C	T	A	G	C
				← start			gene									← stop							← start			gene			← stop			

Goal. Write a Java program to find genes in a given genome.

String client warmup: Identifying a potential gene

Goal. Write a Java program to determine whether a given string is a potential gene.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	T	G	C	A	T	A	G	C	G	C	A	T	A	G

← *start* *gene* *stop* →

```
% java Gene ATGCATAGCGCATAG
true
% java Gene ATGCGCTGCGTCTGTACTAG
false
% java Gene
ATGCCGTGACGTCTGTACTAG
false
```

```
public class Gene
{
    public static boolean isPotentialGene(String dna)
    {
        if (dna.length() % 3 != 0) return false;
        if (!dna.startsWith("ATG")) return false;
        for (int i = 0; i < dna.length() - 3; i+=3)
        {
            String codon = dna.substring(i, i+3);
            if (codon.equals("TAA")) return false;
            if (codon.equals("TAG")) return false;
            if (codon.equals("TGA")) return false;
        }
        if (dna.endsWith("TAA")) return true;
        if (dna.endsWith("TAG")) return true;
        if (dna.endsWith("TGA")) return true;
        return false;
    }
    public static void main(String[] args)
    {
        StdOut.println(isPotentialGene(args[0]));
    }
}
```

String client exercise: Gene finding

Goal. Write a Java program to find genes in a given genome.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
A	T	A	G	A	T	G	C	A	T	A	G	C	G	C	A	T	A	G	C	T	A	G	A	T	G	T	G	C	T	A	G	C
				← start			← gene									← stop							← start			← gene			← stop			

Algorithm. Scan left-to-right through dna.

- If start codon ATG found, set **beg** to index **i**.
- If stop codon found and substring length is a multiple of 3, print gene and reset **beg** to -1.

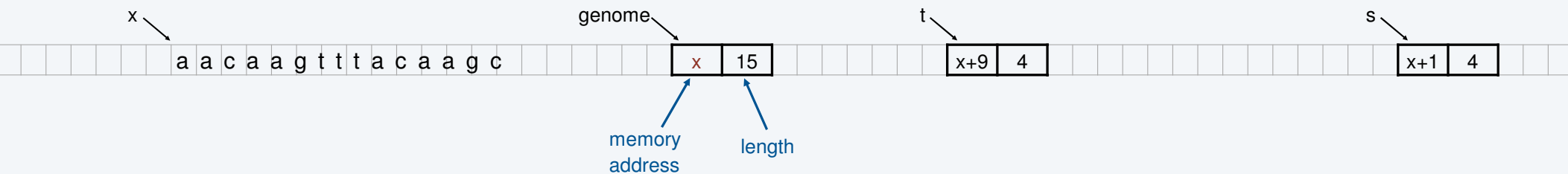
i	codon		beg	output	remainder of input string
	start	stop			
0			-1		ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
1		TAG	-1		TAGATGCATAGCGCATAGCTAGATGTGCTAGC
4	ATG		4		ATGCATAGCGCATAGCTAGATGTGCTAGC
9		TAG	4		TAGCGCATAGCTAGATGTGCTAGC
16		TAG	4	CATAGCGCA	TAGCTAGATGTGCTAGC
20		TAG	-1		TAGATGTGCTAGC
23	ATG		23		ATGTGCTAGC
29		TAG	23	TGC	TAGC

Implementation.
Entertaining
programming
exercise!

OOP context for strings

Possible memory representation of

```
String genome = "aacaagtttacaagc";  
String s = genome.substring(1, 5);  
String t = genome.substring(9, 13);
```



Implications

- s and t are different strings that share the same value "acaa".
- (s == t) is false (because it compares addresses).
- (s.equals(t)) is true (because it compares character sequences).
- Java String interface is more complicated than the API.

Object-oriented programming: summary

Object-oriented programming.

- Create your own data types (sets of values and ops on them).
- Use them in your programs (manipulate *objects*).

← An **object** holds a data type value.
Variable names refer to objects.

In Java, programs manipulate references to objects.

- String, Picture, Color, arrays, (and everything else) are *reference types*.
- Exceptions: boolean, int, double and other *primitive types*.
- OOP purist: Languages should not have separate primitive types.
- Practical programmer: Primitive types provide needed efficiency.



T	A	G	A	T	G	T	G	C	T	A	G	C
---	---	---	---	---	---	---	---	---	---	---	---	---

This lecture: You can write programs to manipulate sounds, colors, pictures, and strings.

Next lecture: You can *define your own abstractions* and write programs that manipulate them.

Image sources

<http://upload.wikimedia.org/wikipedia/commons/6/6a/>

[Construction_Site_for_The_Oaks_High_School_Retford_-_geograph.org.uk_-_89555.jpg](#)

Image sources

<http://archive.hudsonalpha.org/education/outreach/basics/eye-color>

<http://www.designishistory.com/1940/joseph-albers/>

http://en.wikipedia.org/wiki/Josef_Albers#mediaviewer/File:Josef_Albers.jpg

http://fr.freepik.com/photos-libre/oeil-au-beurre-noir-et-blanc_620699.htm

http://en.wikipedia.org/wiki/The_Treachery_of_Images#mediaviewer/File:MagrittePipe.jpg

http://static.tvtropes.org/pmwiki/pub/images/not-a-pipe-piraro_598.png



9. Abstract Data Types

- Overview
- Color
- Image processing
- String processing



INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

9. Abstract Data Types



<http://introcs.cs.rutgers.edu>