8. Abstract Data Types
8. Abstract Data Types

- Overview
- Color
- Image processing
- String processing
Abstract data types

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.
Object-oriented programming (OOP)

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

Examples (stay tuned for details)

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

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
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).

### Operations (API)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String(String s)</td>
<td>create a string with the same value</td>
</tr>
<tr>
<td>int length()</td>
<td>string length</td>
</tr>
<tr>
<td>char charAt(int i)</td>
<td><em>ith</em> character</td>
</tr>
<tr>
<td>String substring(int i, int j)</td>
<td><em>ith</em> through <em>(j-1)st</em> characters</td>
</tr>
<tr>
<td>boolean contains(String sub)</td>
<td>does string contain sub?</td>
</tr>
</tbody>
</table>
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.

```java
String s;
s = new String("Hello, World");
StdOut.println(s.substring(0, 5));
```
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?
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.
**Image sources**

http://upload.wikimedia.org/wikipedia/commons/6/6a/Construction_Site_for_The_Oaks_High_School_Retford_-_geograph.org.uk__89555.jpg
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.

**API (operations)**

<table>
<thead>
<tr>
<th>Color</th>
<th>getRed()</th>
<th>green intensity</th>
<th>getGreen()</th>
<th>blue intensity</th>
<th>getBlue()</th>
<th>brighter()</th>
<th>darker version of this color</th>
<th>toString()</th>
<th>string representation of this color</th>
<th>equals(Color c)</th>
<th>is this color the same as c's?</th>
</tr>
</thead>
<tbody>
<tr>
<td>R (8 bits)</td>
<td>red intensity</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>119</td>
<td>105</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G (8 bits)</td>
<td>green intensity</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>64</td>
<td>33</td>
<td>105</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B (8 bits)</td>
<td>blue intensity</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td>128</td>
<td>27</td>
<td>105</td>
<td></td>
<td></td>
</tr>
<tr>
<td>color</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Albers squares

Josef Albers. A 20th century artist who revolutionized the way people think about color.
Color client example: Albers squares

**Goal.** Write a Java program to generate Albers

% java AlbersSquares 0 64 128 105 105 105

% java AlbersSquares 251 112 34 177 153 71

% java AlbersSquares 28 183 122 15 117 123
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);
    }
}
Computing with color: monochrome luminance

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

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

```java
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

<table>
<thead>
<tr>
<th>color</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>red intensity</td>
<td>255 0 0 0</td>
</tr>
<tr>
<td>green intensity</td>
<td>0 255 0 0</td>
</tr>
<tr>
<td>blue intensity</td>
<td>0 0 255 0</td>
</tr>
<tr>
<td>luminance</td>
<td>76 150 29 0</td>
</tr>
</tbody>
</table>

Applications (next)
- Choose colors for displayed text.
- Convert colors to grayscale.
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.

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

```java
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**

<table>
<thead>
<tr>
<th>examples</th>
<th>red intensity</th>
<th>255 0 0 0 255 0 119 105</th>
</tr>
</thead>
<tbody>
<tr>
<td>green intensity</td>
<td>0 255 0 0 255 64 33 105</td>
<td></td>
</tr>
<tr>
<td>blue intensity</td>
<td>0 0 255 0 255 128 27 105</td>
<td></td>
</tr>
<tr>
<td>color</td>
<td></td>
<td>[image showing color samples]</td>
</tr>
<tr>
<td>luminance</td>
<td>76 150 29 0 255 52 58 105</td>
<td></td>
</tr>
<tr>
<td>grayscale</td>
<td></td>
<td>[image showing grayscale samples]</td>
</tr>
</tbody>
</table>
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

```java
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.

Java. These are not colors.

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

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

Surrealist computer scientist: Neither is this.

% java RandomSeq 10000 java Average
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://static.tvropes.org/pmwiki/pub/images/not-a-pipe-piraro_598.png
9. Abstract Data Types

- Overview
- Color
- Image processing
- String processing
**Picture ADT**

A Picture is a 2D array of pixels.

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

### API (operations)

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

### Values (2D arrays of Colors)

- **pixel**
- **width**
- **height**
- **row**
- **column**

A Picture is a 2D array of pixels defined in terms of its ADT values (typical).
Picture client example: Grayscale filter

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

Source: mandrill.jpg

% java Grayscale mandrill.jpg
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();
    }
}
Pop quiz 1a on image processing

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

```java
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)?

```java
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)?

```java
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)?

```java
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?

```java
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?

```java
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.
Goal. Write a Java program to scale an image (arbitrarily and independently on $x$ and $y$).
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.
**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.
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();
    }
}
More image-processing effects

RGB color separation

swirl filter  wave filter  glass filter  Sobel edge detection
CS.8.C.ADTs.Images
9. Abstract Data Types

- Overview
- Color
- Image processing
- String processing
String ADT

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

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

### Operations (API)

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

Defined in terms of its ADT values (typical)
Programming with strings: typical examples

Is the string a palindrome?

```java
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;
}
```

Find lines containing a specified string in StdIn

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

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

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

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.

```
ATG C A T A G C G C A T A G

start       gene     stop
```

% java Gene ATGCATAGCCCATAG
true
% java Gene ATGCGCTGCGTCTGTACTAG
false
% java Gene ATGCCGTGACTGCTACTAG
false

```java
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.

![DNA String](image)

**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.

<table>
<thead>
<tr>
<th>i</th>
<th>codon</th>
<th>beg</th>
<th>output</th>
<th>remainder of input string</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TAG</td>
<td>-1</td>
<td>ATAGATGCATAGCGCATAGCTAGATGCTAGC</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TAG</td>
<td>-1</td>
<td>TAGATGCATAGCGCATAGCTAGATGCTAGC</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ATG</td>
<td>4</td>
<td>ATGCATAGCGCATAGCTAGATGCTAGC</td>
<td>CATAGCGCA</td>
</tr>
<tr>
<td>9</td>
<td>TAG</td>
<td>4</td>
<td>TAGCGCATAGCTAGATGCTAGC</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>TAG</td>
<td>4</td>
<td>ATGCATAGCGCATAGCTAGATGCTAGC</td>
<td>TAGCTAGATGCTAGC</td>
</tr>
<tr>
<td>20</td>
<td>TAG</td>
<td>-1</td>
<td>TAGATGCTAGCTAGC</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>ATG</td>
<td>23</td>
<td>ATGCATAGCGCATAGCTAGATGCTAGC</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>TAG</td>
<td>23</td>
<td>TGC</td>
<td>TAGCTAGATGCTAGC</td>
</tr>
</tbody>
</table>
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).

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.

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.
CS.8.D.ADTs.Strings
8. Abstract Data Types