3.1 Using Data Types

Data Types

Data type. Set of values and operations on those values.

Primitive types. Values directly map to machine representation; ops directly map to machine instructions.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Set of Values</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>true, false</td>
<td>not, and, or, xor</td>
</tr>
<tr>
<td>int</td>
<td>-2^31 to 2^31 - 1</td>
<td>add, subtract, multiply</td>
</tr>
<tr>
<td>double</td>
<td>any of 2^64 possible reals</td>
<td>add, subtract, multiply</td>
</tr>
</tbody>
</table>

We want to write programs that process other types of data.
- Colors, pictures, strings, input streams, ...
- Complex numbers, vectors, matrices, polynomials, ...
- Points, polygons, charged particles, celestial bodies, ...

Objects

Object. Holds a data type value; variable name refers to object.

Object-oriented programming.
- Create your own data types (set of values and ops on them).
- Use them in your programs (manipulate objects that hold values).

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Set of Values</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>24 bits</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>

This lecture. Use existing data types.
Next lecture. Create your own data types.

Constructors and Methods

To construct a new object:
- Use keyword new (to invoke constructor).
- Use name of data type (to specify which type of object).

To apply an operation:
- Use name of object (to specify which object).
- Use the dot operator (to invoke method).
- Use the name of the method (to specify which operation).

```
// declare a variable (object name): declare a construction to create an object
String s;
String[] s;
String s = new String("Hello, world!");

// call a method that operates on the object value
s = s.substring(0, 5);

System.out.println(s);
```

Image Processing
### Color Data Type

#### Color. A sensation in the eye from electromagnetic radiation.

**Set of values.** [RGB representation] 256³ possible values, which quantify the amount of red, green, and blue, each on a scale of 0 to 255.

**API.** Application Programming Interface.

```
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 0.299 * r + 0.587 * g + 0.114 * b;
    }
}
```

#### Color Compatibility

Q. Which font colors will be most readable with which background colors on computer and cell phone screens?

A. Rule of thumb: difference in luminance should be \( \geq 128 \).

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

### Monochrome Luminance

#### Monochrome luminance. Effective brightness of a color.

**NTSC formula.**

\[ Y = 0.299 \times r + 0.587 \times g + 0.114 \times b. \]

```
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 0.299 * r + 0.587 * g + 0.114 * b;
    }
}
```

### Grayscale

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

**Convert to grayscale.** Use luminance to determine value.

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

### OOP Context for Color

#### Possible memory representation.

```
255 0 255 0 0 105 105 105
```

Object reference is analogous to variable name.

- We can manipulate the value that it holds.
- We can pass it to (or return it from) a method.

---

**Color.** A sensation in the eye from electromagnetic radiation.

**Set of values.** [RGB representation] 256³ possible values, which quantify the amount of red, green, and blue, each on a scale of 0 to 255.

**API.** Application Programming Interface.

```
public class Color {
    int red, green, blue;
    public Color(int r, int g, int b) {
        red = r;
        green = g;
        blue = b;
    }
    public int getRed() { return red; }
    public int getGreen() { return green; }
    public int getBlue() { return blue; }
    public Color brighter() { return new Color(red + 1, green + 1, blue + 1); }
    public Color darker() { return new Color(red - 1, green - 1, blue - 1); }
    public String toString() {
        return "Color(\d+, \d+, \d+)";
    }
}
```

**API.** Application Programming Interface.

```
public class Color {
    int red, green, blue;
    public Color(int r, int g, int b) {
        red = r;
        green = g;
        blue = b;
    }
    public int getRed() { return red; }
    public int getGreen() { return green; }
    public int getBlue() { return blue; }
    public Color brighter() { return new Color(red + 1, green + 1, blue + 1); }
    public Color darker() { return new Color(red - 1, green - 1, blue - 1); }
    public String toString() {
        return "Color(\d+, \d+, \d+)";
    }
}
```

**API.** Application Programming Interface.

```
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 0.299 * r + 0.587 * g + 0.114 * b;
    }
}
```

**API.** Application Programming Interface.

```
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 0.299 * r + 0.587 * g + 0.114 * b;
    }
}
```
References

René Magritte. “This is not a pipe.”

Java. This is not a color.

Color sienna = new Color(160, 82, 45);
Color c = sienna.darker();

OOP. Natural vehicle for studying abstract models of the real world.

This is Not a Pipe

Neither is this.

This is Not a Pipe

Dan Piraro,
http://www.uexpress.com

% java RandomSeq 10000 | java Average

Picture Data Type

Raster graphics. Basis for image processing.

Set of values. 2D array of color objects (pixels).

API

public class Picture

create a picture from a file
create a blank w by h picture
return the width of the picture
return the height of the picture
set the color of pixel (x, y) to c
set the color of pixel (x, y) to c
display the image to a window
save the image to a file

Image Processing: Grayscale Filter

Goal. Convert color image to grayscale according to luminance formula.

import java.awt.Color;

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

Image Processing: Grayscale Filter

import java.awt.Color;

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

Image Processing: Scaling Filter

Goal. Shrink or enlarge an image to desired size.

Downscaling. To shrink, delete half the rows and columns.
Upscaling. To enlarge, replace each pixel by 4 copies.
Goal. Shrink or enlarge an image to desired size.

Uniform strategy. To convert from \(w_s\)-by-\(h_s\) to \(w_t\)-by-\(h_t\):
- Scale column index by \(w_s / w_t\).
- Scale row index by \(h_s / h_t\).
- Set color of pixel \((x, y)\) in target image to color of pixel 
  \((x \times w_s / w_t, y \times h_s / h_t)\) in source image.

Java code example:

```java
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.ImageView;
import java.awt.RenderingHints;
import java.awt.Transparency;
import java.awt.image.BufferedImage;
import java.awt.image.Raster;
import java.awt.image.WritableRaster;
import java.io.File;
import java.io.IOException;

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]);

        Image source = ImageIO.read(new File(filename));
        BufferedImage target = new BufferedImage(w, h, Transparency.TRANSLUCENT);

        for (int tx = 0; tx < target.getWidth(); tx++) {
            for (int ty = 0; ty < target.getHeight(); ty++) {
                int sx = tx * source.getWidth() / target.getWidth();
                int sy = ty * source.getHeight() / target.getHeight();

                Color color = source.get(sx, sy);
                target.set(tx, ty, color);
            }
        }

        ImageView sourceView = new ImageView(source);
        sourceView.show();
        ImageView targetView = new ImageView(target); // scale and display
        targetView.show();
    }
}
```

Scaling filter. Creates two Picture objects and two windows.

More Image Processing Effects

String Data Type

String data type. Basis for text processing. Set of values: Sequence of Unicode characters.

API:

```java
public class String {
    public static String[] split(String str, String delimiter)
    // returns an array of substrings that are separated by the specified delimiter
}
```
Typical String Processing Code

```java
public static boolean equals(String x) {
    String s = x.toLowerCase();
    for (int i = 0; i < x.length(); i++)
        if (s.charAt(i) != e.charAt(i))
            return false;
    return true;
}
```

String s = genome.substring(1, 5);
String t = genome.substring(9, 13);
if (s.equals(t))
    return true;

```
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.
- Preceded by \texttt{ATG}, \texttt{CAT}, or \texttt{GCT}.
- Multiple of 3 nucleotides.
- Succeeded by \texttt{TAG}, \texttt{TAA}, or \texttt{TGA}.

Goal. Find all genes.

```

Gene Finding: Algorithm

Algorithm. Scan left-to-right through genome.
- If start codon, then set beg to index i.
- If stop codon and substring is a multiple of 3, output gene.

```

```
Gene Finding: Implementation

public class GeneFind {
    public static void main(String[] args) {
        String start = args[0];
        String stop = args[1];
        String genome = StdIn.readString();
        int beg = -1;
        for (int i = 0; i < genome.length(); i++)
            if (genome.substring(i).equals(start))
                beg = i;
            if (beg > -1 && genome.substring(i).equals(stop))
                StdOut.println(beg + " to " + (i + 1));
    }
}
```

OOP Context for Strings

```
Possible memory representation of a string.
- genome = "aacaagtttacaagc"

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>G</td>
<td>T</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
</tbody>
</table>
```

```
Possible memory representation of a string.
- genome = "aacaagtttacaagc"

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>g</td>
<td>t</td>
<td>t</td>
<td>c</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
</tbody>
</table>
```

```java
String genome = StdIn.readString();
String s = genome.substring(1, 5);
String t = genome.substring(9, 13);
```
In and Out

Bird’s Eye View (Revisited)

Non-Standard Input

Screen Scraping

Day Trader

Screen Scraping

Day Trader

Non-Standard Input

Standard input. Read from terminal window.

Goal. Read from several different input streams.

Ex: Are two text files identical?

```java
public class Diff {
    public static void main(String[] args) {
        In in0 = new In(args[0]); // read from one file
        In in1 = new In(args[1]); // read from another file
        String s = in0.readLine();
        String t = in1.readLine();
        StdOut.println(s.equals(t));
    }
}
```

or use OS to redirect from one file

or use OS to redirect from one file

Read raw html from http://finance.yahoo.com/q?s=goog

```java
public class StockQuote {
    public static void main(String[] args) {
        String name = "http://finance.yahoo.com/q?s=";
        In in = new In(name + args[0]);
        String input = in.readLine();
        int start = input.indexOf("Last Trade:");
        int from = input.indexOf("<b>", start);
        int to = input.indexOf("</b>", from);
        String price = input.substring(from + 3, to);
        StdOut.println(price);
    }
}
```

Please, please use at your own financial risk.

Warning. Please, please use at your own financial risk.

Google searches the web for "Day Trader" and finds

http://finance.yahoo.com/q?s=goog

Add bells and whistles.

- Plot price in real-time.
- Notify user if price dips below a certain price.
- Embed logic to determine when to buy and sell.
- Automatically send buy and sell orders to trading firm.

The New Yorker, September 6, 1999
OOP Summary

Object. Holds a data type value; variable name refers to object.

In Java, programs manipulate references to objects.
- Exception: primitive types, e.g., boolean, int, double.
- Reference types: String, Picture, Color, arrays, everything else.
- OOP purist: language should not have separate primitive types.

Bottom line. We wrote programs that manipulate colors, pictures, and strings.

Next time. We’ll write programs that manipulate our own abstractions.

Extra Slides

Color Separation

```java
import java.awt.Color;
public class ColorSeparation {
    public static void main(String args[]) {
        int width = pic.width();
        int height = pic.height();
        Picture R = new Picture(width, height);
        Picture G = new Picture(width, height);
        Picture B = new Picture(width, height);
        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                Color c = pic.get(x, y);
                int r = c.getRed();
                int g = c.getGreen();
                int b = c.getBlue();
                R.set(x, y, new Color(r, 0, 0));
                G.set(x, y, new Color(0, g, 0));
                B.set(x, y, new Color(0, 0, b));
            }
        }
        R.show();
        G.show();
        B.show();
    }
}
```

Color Separation.java. Creates three Picture objects and windows.

Memory Management

Value types.
- Allocate memory when variable is declared.
- Can reclaim memory when variable goes out of scope.

Reference types.
- Allocate memory when object is created with new.
- Can reclaim memory when last reference goes out of scope.
- Significantly more challenging if several references to same object.

Garbage collector. System automatically reclaims memory; programmer relieved of tedious and error-prone activity.