CSE 1710

Lecture 10

Image and Pixel Services

The String class

Strings

We have covered two chunks of material:

Week 1:

- "String Literals" pp. 22-23; Fig 1.12; PT 1.8

Week 6:

- "The String Class" Section 6.1.1, pp. 219-220
- "The Masquerade and the + Operator" Section 6.1.2, pp. 221-224

What is the `String` class?

- provides services to represent and perform tasks on strings
- the class encapsulates a string as a ***sequence*** of *Unicode characters*
- compare/contrast the representation of a string with the representation of a character

	String	**Character**
type:	`String`	`char`
	non-primitive	primitive
operators:	+	arithmetic operators
		(cast to int)

The + Operator : Predict the Output

```
String x = "hi\n";
String y = "there";
String z = x + y;
output.println(z);

char a = 'H';
char b = 'I';
output.println(a+b);
```

Details about the character sequence:

- the sequence is indexed
  - the ***first*** position is index “0”
  - the ***final*** position is index “the length of the sequence minus 1”

- `String` provides services to tell us about the sequence
- Methods include:
  - `int : length()`
  - `char : charAt(int)`

- what if index is out of bounds?

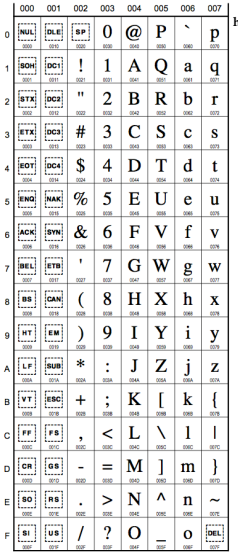
Unicode

- a *unicode character*:
  - is a non-negative numeric value
  - has a corresponding character according to the Unicode character tables (as defined by the Unicode Consortium)

- Unicode is a computing industry standard
  - provides consistent encoding, representation and handling of text
  - text as expressed in most of the world's writing systems
  - used by Java and many other programming languages

The letter J is found in column ‘004’ and row ‘A’, which makes ‘004A’

This is a hexadecimal number, denoted \u004A



http://unicode.org/charts/PDF/U0000.pdf

To convert a Unicode hexadecimal number to decimal:

1. take the hex number and identify the four digits:
   \u004A ➔ $\mathbf{d_3d_2d_1d_0}$ ➔ 0 0 4 A
2. Convert each hex digit to decimal:
   - the hex d_i span the digits: [0,…, 9, A, B, C, D, E, F]
   - this maps to the decimal digits: [0, 15]
   - hex ‘A’ maps to decimal ’10’, …, ‘F’ maps to ’15’
3. Plug the digits into the following formula:

$$\mathbf{d_3d_2d_1d_0 = d_3\times16^3 + d_2\times16^2 + d_1\times16^1 + d_0\times16^0}$$

Example: so to convert \u004A to decimal:

$= 0 \times 163 + 0 \times 162 + 4 \times 161 + 10 \times 160$

$= 4 \times 16 + 10 \times 1$

$= 64 + 10 = 74$

Unicode

- The Unicode Standard consists of a repertoire of more than 109,000 characters covering 93 scripts
  - Cyrillic, Latin, Bengali, Thai, Greek, …
  - the basic set is “Controls and Basic Latin”
  - U000.pdf, also see Appendix A of JBA
- Unicode value denoted \uXXXX, where XXXX is a hexadecimal value
  - the decimal value 15 is represented as \u000F
- unicode makes is possible to talk about the *distance* between two characters

How to use Unicode directly

```
String s6 = "\u00A5";
String s7 = "\u00A7";
String s8 = "\u00AE";
String s9 = "\u2C16";

char c6 = '\u00A5';
char c7 = '\u00A7';
char c8 = '\u00AE';
char c9 = '\u2C16';
```

In the case of \u2C16 *we may be able to represent a unicode character, but PrintStream may not be able to print it to the console…*

How to use Unicode directly

```
String s6 = "\u00A5";
String s7 = "\u00A7";
String s8 = "\u00AE";

char c6 = '\u00A5';
char c7 = '\u00A7';
char c8 = '\u00AE';
```


One Caveat:

some Unicode characters cannot be ***printed*** to the console.

For example, \u2C16 is taken from the “Glogolitic” table and represents the character:

We can represent the character, but PrintStream cannot print it to the console…

```
String s9 = "\u2C16";
```

REMEMBER!

– Any string is represented by **an object**

– A variable of type `String` is used to store **the address** of the object.

– The `String` object has a **state**
  - the **state** of an object is defined as **the value of all its attributes**
  - the **only attribute** of a `String` object is the attribute that represents the sequence of characters
  - the state of a `String` object basically boils down to **what is its sequence of characters?**

REMEMBER!

– If the state of a `String` object is such that its **sequence has no characters at all,** how do we understand this?
  - this is the *empty string*
  - the string has length **zero**
  - ***THIS IS NOT A NULL STRING***

  - **What is this "null string"?**
    - technically speaking, "null string" is not really a correctly-formed term, there is no such thing
    - HOWEVER, it is often used to mean a **string reference** that is set to `null`.
    - This means that a String reference has been declared, but that there is NO `String` object.

Can we modify the state of a `String` object?

– NO

– Once a string object is created, it cannot be changed.
  - This is called *immutability*
  - Strings are *immutable*

– This is an unusual property – MOST other objects are mutable

But what if we need to modify the state of a `String` object?

Instead of modifying the sequence, we just create new strings that are modified verisons of the originals.

– It is fast and easy, thanks to the + operator

– Given this, is it correct to say that String has mutators?
  - not technically; they are actually *generators of new modified objects*

Images

This material will be presented in lecture.

Take good notes – there is little material in the textbook

To work with images, we need to:

1. work with the file system
2. work with the operating system's window manager and the platform's graphics hardware
3. understand colour models and image representation formats
4. understand the services of `Pixel` and `Picture` classes
5. iterate and construct conditions [later in course]

About files…
pathnames are system dependent

- Windows Local File System (LFS):
  - `C:\USER\DOCS\LETTER.TXT`
- Windows Uniform Naming Convention (UNC)
  - `\\Server\Volume\File`
- Unix-like OS
  - `/home/user/docs/Letter.txt`

ABSTRACTION:

Which details are system dependent?

What can be abstracted away?

also **lists** of pathnames are system dependent

- Windows Local File System (LFS):
  - `C:\USER\DOCS\;C:\BIN`
- Unix-like OS
  - `/home/user/docs/:/usr/bin/:/sbin/`

ABSTRACTION:

Which details are system dependent?

What can be abstracted away?

Pathname abstraction

ABSTRACTION:
Which details are system dependent?
What can be abstracted away?

- separator (e.g., `/`, `\`)
- system prefix (e.g., `/`, `\\`, `C:\`)
- path separator (e.g., `;`, `:`)

Useful class: `java.io.File`

- not a utility class; encapsulates `File` objects
  - a *file* in this context can be
    - a directory
    - a "normal file" (defined as something that is **not** a directory)
  - The constructor for `File` objects requires a *pathname*
- The class `File` provides **static** features
  - representation of the system-dependent elements
    - separator, path separator
    - demo: `L10_App2`

The encapsulation of a `File`…

- provides delegation of file-related tasks:
  - does **this** file exist?
  - is **this** file a directory or a normal file?
  - can I write to **this** file?
  - which files are in **this** directory, if any?
    - assumes this file is a directory
  - make a directory, as specified by **this** file
    - assumes pathname is not already in use and operation is allowed

The encapsulation of a `File`…

- **does not** provide the means to *write* to the file object ☹
  - for this, you need the services of `FileWriter`
  - a `FileWriter` object encapsulates all of the working of writing content to a `File` object
  - defer this aspect for the time being

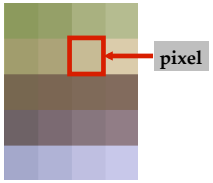
How do I get my hands on a `File` object?

- **construct** one from scratch
  - `L10_App3`
- let the user **specify** one for you
  - `L10_App4`

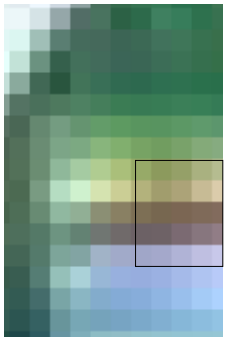
Digital Images

- storage
  - files contain *pixel* and/or *vector* data
    - pixel – a single point at a given coordinate that has specific colour attributes
    - vector data – information about graphic primitives, such as lines, curves, shapes
      - e.g., "draw a circle with radius r with center point at location (x,y) and with a solid black stroke and a solid fill"
- display
  - whatever the file format, the file is *rasterized* to pixels for the graphic display

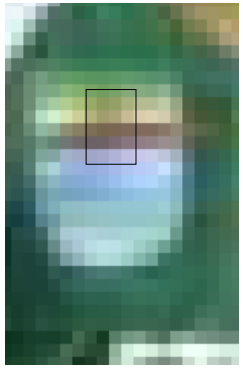
A Pixel Image



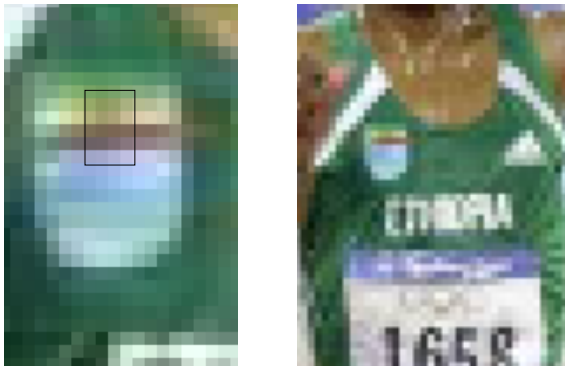
A Pixel Image



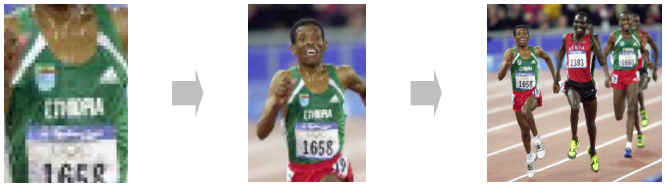
A Pixel Image



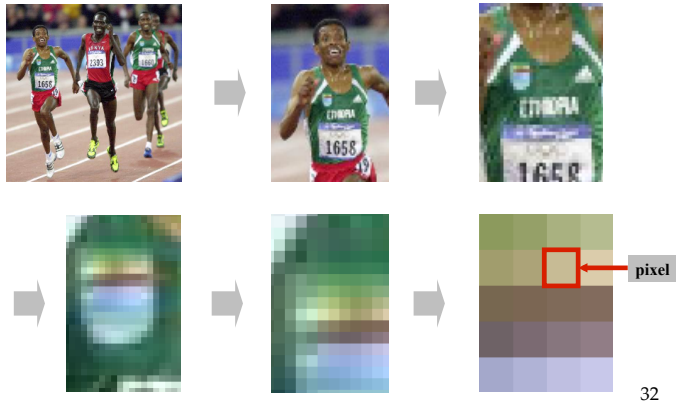
A Pixel Image



A Pixel Image



A Pixel Image



Raster

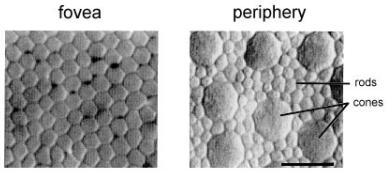
- a rectangular grid of pixels
- each element has a (*x*,*y*) coordinate
  - the convention is that (0,0) is in the upper left hand corner
  - the *x* part of coordinate indicates the column
  - the *y* part of the coordinate indicates the row
  - ***in*** the door and ***down*** the stairs
- `L8App4` – demo of picture explorer

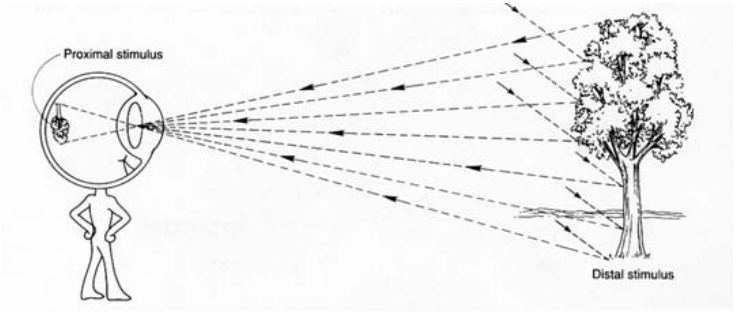
What is the RGB model? Why is it intuitive?

- First, we will discuss the basics of vision…
- the **retina** of the human eye is the location of the photoreceptors
  - *rods*
  - *cones*

Areas of the Retina

- the center, the *fovea*
  - only cone receptors, tightly packed
    - three types of cones: short-, medium-, and long-wavelength
  - no rods
- periphery of retina
  - proportion of rods to cones increase toward edge of retina



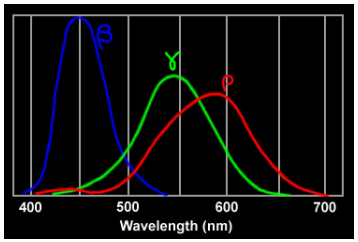


Foveal vision

– fovea has a concentration of three types of cones
– each type is attuned to a different wavelength

Hue

Red –
perceived by long-wavelength cones
Green –
perceived by medium-wavelength cones
Blue –
perceived by short-wavelength cones



Specialized photoreceptors

– peripheral vision
  • contains mostly rods
  • rods are attuned to a broad spectrum of light
    – not specialized to particular wavelengths
    – more sensitive than cones (the threshold is lower)
– fovea
  • specialized for acute detailed vision
– periphery
  • does not provide acuity, but does detect change in scene (e.g., movement)
  • something happened, but not what
  • rods are more sensitive to light than cones

Colour is complicated

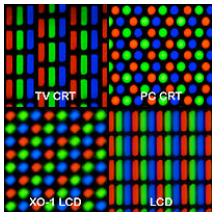
– perception based on 2 types of receptors (hue and intensity)
– our brain does more seeing than our eyes
– what we call colour is more accurately described as hue and brightness

A Key Fact

– the combination of red, blue and green is indistinguishable from **white** to the human eye
– this is exploited by computer displays

Pixels and Subpixels

- Many displays have a cluster of R, G, B sub-pixels for each pixel



- max *intensity* for R, G, B = seen as white
- min *intensity* for R, G, B = seen as black
- … and other saturated colours…

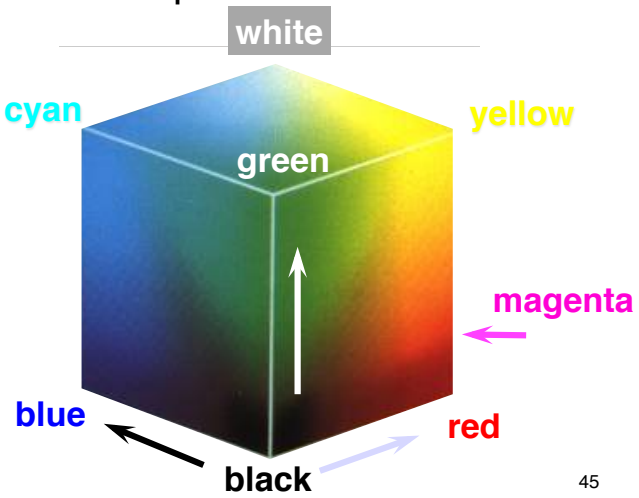
Color	Red	Green	Blue
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Yellow	255	255	0
Cyan	0	255	255
Magenta	255	0	255
White	255	255	255
Black	0	0	0

Other cases…

- Intensities are all the same
  – perceived as shade of grey
- Intensities are different
  – perception depends on relative difference between strongest and weakest intensities

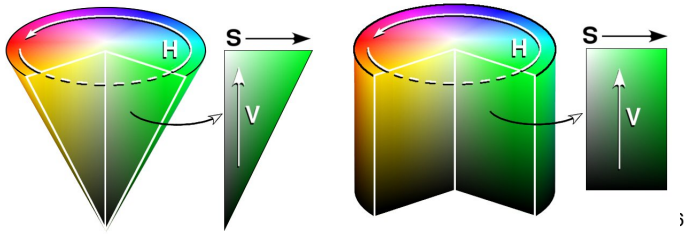
- Given a colour, it can be difficult to determine the RGB values without a colour chooser

RGB Colour Space



Hue-Saturation-Value (HSV) Model

- Each of hue, saturation, and brightness individually specified
- similarities to the way humans perceive and describe colour



The Picture Explorer

- `L10_App5`

What is this *window manager* and why do I care?

- first, a more fundamental question:
  - what is the *desktop metaphor*?
    - a set of UI concepts that treat the computer display as if it were the user's real-world desktop
    - desktop items include: documents, folders, desk accessories (calculator, calendar)
    - the purity of metaphor now diluted and now includes things without real-world counterpart
      - » menu bars, task bars, docks, trashcans,
- key feature: desktop items can ***overlap***

What is this *window manager* and why do I care?

- it is system software
  - operates computer hardware (the graphics card, in this case)
  - provides platform for running apps
- it provides display functionality for apps
  - controls placement and appearance of windows
    - open, close, minimize, maximize, move, resize
  - implements look and feel of window decorators
    - borders (decorative and functional), titlebar (title and buttons)

The window manager provides services to the VM

- **VM:** *Hi WM, I have this app that wants to draw something graphical on the display…*
- **WM:** *ok VM, here is some screen real estate.*
  - Your app can draw within that region, but not outside it. *(It can try, but I will never permit it to happen)*
  - I will decide what actually gets drawn. *(There may be overlapping windows, so your real estate may be occluded)*
  - I can't guarantee this region. *(The user may move the window, or resize or minimize it)*