Overview

- Binary representation
- Image representation
- Java Applets
- Data types revisited

Data Representation

- Computers store all information digitally, using binary codes:
  - numbers
  - text
  - images
  - audio
  - video
  - program instructions

Why Binary Numbers?

- Simplest way to represent digital information:
  - Electronic circuits: high/low voltage
  - Magnetic devices (eg hard drive): positive/negative
  - Optical devices (eg DVD): light reflected/not reflected due to microscopic grooves

A binary digit is called a *bit* - binary digit
A *byte* is a group of eight bits

- a number?
- a letter?
- the red component of a pixel?
- a program instruction?

Computing devices store & use binary codes to represent data of **all kinds**

A *binary code* is a string of bits. A 1 bit code has 2 possibilities (0, 1); a 2 bit code has 4 possibilities (00, 01, 10, 11); a 3 bit code has 8 possibilities (000, 001, 010, 011, 100, 101, 110, 111); and so on.

Each additional bit doubles the number of possible codes.

<table>
<thead>
<tr>
<th>1 bit</th>
<th>2 bits</th>
<th>3 bits</th>
<th>4 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>001</td>
<td>0001</td>
</tr>
<tr>
<td>10</td>
<td>010</td>
<td>010</td>
<td>0010</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>011</td>
<td>0011</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>10</td>
<td>0100</td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>11</td>
<td>0101</td>
</tr>
<tr>
<td>110</td>
<td></td>
<td>110</td>
<td>001</td>
</tr>
<tr>
<td>111</td>
<td></td>
<td>111</td>
<td>001</td>
</tr>
</tbody>
</table>

Each additional bit doubles the number of possible codes.

**Storage requirements examples**

- If a code requires 5 bits, a document consisting of 4000 such codes will require a total of: 5 x 4000 = 20,000 bits
  - How many **bytes** is that? __________

- If a code requires 8 bits (i.e., a byte), a document consisting of 2000 such codes will require 2000 bytes.
  - How many **bits** is that? __________

- If a code requires 32 bits, a program that needs to store 2000 such codes will require __________ bits or __________ bytes.
Storage Capacity

- Every memory device has a storage capacity, indicating the number of bytes it can hold.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Symbol</th>
<th>Number of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>kilobyte</td>
<td>KB</td>
<td>$2^{10} = 1024$</td>
</tr>
<tr>
<td>megabyte</td>
<td>MB</td>
<td>$2^{20}$ (over one million)</td>
</tr>
<tr>
<td>gigabyte</td>
<td>GB</td>
<td>$2^{30}$ (over one billion)</td>
</tr>
<tr>
<td>terabyte</td>
<td>TB</td>
<td>$2^{40}$ (over one trillion)</td>
</tr>
<tr>
<td>petabyte</td>
<td>PB</td>
<td>$2^{50}$ (a whole bunch)</td>
</tr>
</tbody>
</table>

Overview

- Binary representation
- Image representation
- Java Applets
- Data types revisited

What’s a picture?

- Programs represent pictures as grids of picture elements or pixels.

Representing Images

- Bitmap: 1 bit
- Grayscale: 8 bits
- RGB Color: 3 colors: red, green, blue; 8 bits/color; 24 bits
Example: Representing Pixels

Example: Representing Pixels

Additive/Subtractive Color

Additive/Subtractive Color

Additive primaries
- combining light
Red Green Blue

Subtractive primaries
- combining ink, thus subtracting light
Cyan Yellow Magenta

Bitmap image

Bitmap image

Grayscale Image

Grayscale Image

Bitmap

1 bit/pixel

Grayscale

8 bits/pixel

A 2000 by 1000 bitmap image contains
___________ pixels.
Thus, we need __________ bits to
to represent this image, or __________ bytes

A 2000 by 1000 grayscale image contains
___________ pixels.
Thus, we need __________ bytes
to represent this image, or __________ bits
RGB Image

A 2000 by 1000 RGB image contains
__________ pixels.

Thus, we need __________ bytes
to represent this image, or __________ bits.

Graphics and images in Java

- Images are represented as objects in Java.
- Color components and positions for pixels can be represented as integers (but also in other ways).
- Java libraries allow flexibility, providing many alternative ways of representing and processing images.
- We will be using the `Graphics` class of the `awt` package and `Japplet` from the `swing` package.

Overview

- Binary representation
- Image representation
- Java Applets
- Data types revisited

Applets

- A Java *applet* is a program that is intended to be transported over the Web and executed using a web browser.
- An applet doesn't have a main method
  - The type of program we have seen so far is a Java *application* - a stand-alone program with a main method.
import javax.swing.JApplet;
import java.awt.*;
public class Einstein extends JApplet {
    public void paint (Graphics page) {
        page.drawRect (50, 50, 40, 40); // square
        page.drawRect (60, 80, 225, 30); // rectangle
        page.drawOval (75, 65, 20, 20); // circle
        page.drawLine (35, 60, 100, 120); // line
        page.drawString("Out of clutter, find simplicity.", 110, 70);
        page.drawString("-- Albert Einstein", 130, 100);
    }
}

Example: Einstein.java

//********************************************************************
// Einstein.java Author: Lewis/Loftus
// Demonstrates a basic applet.
//********************************************************************
import javax.swing.JApplet;
import java.awt.*;
public class Einstein extends JApplet {
    // Draws a quotation by Albert Einstein among some shapes.
    //-----------------------------------------------------------------------------
    public void paint (Graphics page) {
        page.drawRect (50, 50, 40, 40); // square
        page.drawRect (60, 80, 225, 30); // rectangle
        page.drawOval (75, 65, 20, 20); // circle
        page.drawLine (35, 60, 100, 120); // line
        page.drawString("Out of clutter, find simplicity.", 110, 70);
        page.drawString("-- Albert Einstein", 130, 100);
    }
}

CSC 1051 M.A. Papalaskari, Villanova University

Drawing a Line

X
Y
10 150
20 45
page.drawLine (10, 20, 150, 45);
page.drawLine (150, 45, 10, 20);

CSC 1051 M.A. Papalaskari, Villanova University

Drawing a Rectangle

X
Y
50 100
20 40
page.drawRect (50, 20, 100, 40);
Data Representation and Applets

Drawing an Oval

```
x
bounding rectangle

page.drawOval (175, 20, 50, 80);
```

Drawing an Arc

- An arc is defined by an oval, a start angle, and an arc angle:

```
page.drawArc (150, 10, 80, 30, 20, 90);
```

Filled vs unfilled shapes

- Instead of using drawRect(), drawOval() etc, we can use fillRect(), fillOval() etc
- We can set the color using setColor()
  - See Snowman.java
  - See also Snowman applet on a webpage

```java
import javax.swing.JApplet;
import java.awt.*;

public class Snowman extends JApplet {
    public void paint (Graphics page) {
        final int MID = 150;
        final int TOP = 50;
        setBackground (Color.cyan);
        page.setColor (Color.blue);
        page.fillRect (0, 175, 300, 50); // ground
        page.setColor (Color.yellow);
        page.fillOval (-40, -40, 80, 80); // sun
        // continued
    }
}
```
The Java Color Class

- A color in a Java program is represented as an object created from the `Color` class.

- The `Color` class also contains several predefined colors, e.g.
  
<table>
<thead>
<tr>
<th>Object</th>
<th>RGB Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color.black</td>
<td>0, 0, 0</td>
</tr>
<tr>
<td>Color.blue</td>
<td>0, 0, 255</td>
</tr>
<tr>
<td>Color.cyan</td>
<td>0, 255, 255</td>
</tr>
<tr>
<td>Color.orange</td>
<td>255, 200, 0</td>
</tr>
<tr>
<td>Color.white</td>
<td>255, 255, 255</td>
</tr>
<tr>
<td>Color.yellow</td>
<td>255, 255, 0</td>
</tr>
</tbody>
</table>

- Using a color: `page.setColor(Color.blue);`
- Creating a new color: `Color salmon = new Color(255, 140, 128); page.setColor(salmon);`

Translation of programs into machine code

Java Translation is different
The HTML applet Tag
• An applet is embedded into an HTML file using a tag that references the bytecode file of the applet
• The bytecode version of the program is transported across the web and executed by a Java interpreter that is part of the browser.

<html>
  <head>
    <title>The Einstein Applet</title>
  </head>
  <body>
    <applet code="Einstein.class" width=350 height=175>
    </applet>
  </body>
</html>

Overview
• Binary representation
• Image representation
• Java Applets
• Data types revisited

Numeric Primitive Data
• The difference between the numeric primitive types is their size and the values they can store:

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>&lt;-9 x 10^15</td>
<td>&gt; 9 x 10^15</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>+/- 3.4 x 10^38 with 7 significant digits</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>+/- 1.7 x 10^30 with 15 significant digits</td>
<td></td>
</tr>
</tbody>
</table>

Automatic type conversion
Values of different types can be combined in an assignment or an expression
• Example:
  ```java
  int dollars = 5;
  double money = dollars + 2.50;
  System.out.println(dollars + " dollars");
  ```
  • These are all examples of widening conversions, i.e., “smaller” data type ➔ “larger” data type
Converting from one type to another

- **Widening conversions**
  - "small" data type \( \rightarrow \) "larger" one
  - eg: \( \text{int} \rightarrow \text{double} \)
    - 32 bits \( \rightarrow \) 64 bits

- **Narrowing conversions**
  - "large" data type \( \rightarrow \) "smaller" one
  - eg: \( \text{double} \rightarrow \text{int} \)
    - 64 bits \( \rightarrow \) 32 bits
  - narrowing conversions can lose information!
  - narrowing conversions cannot happen automatically
    (for example, through assignment)

Casting

- **Casting** forces a change of type, even if information is lost
- Can be used for both widening and narrowing conversion
- To cast, put the type in parentheses in front of the value to be converted:

```
int total = 5;
double result = (double) total / 2;
int answer = (int) result + 4;
int x = (int) (Math.cos(angle) * 300);
```

Data Conversion

<table>
<thead>
<tr>
<th>Widening Conversions</th>
<th>Narrowing Conversions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>From</strong></td>
<td><strong>To</strong></td>
</tr>
<tr>
<td>byte</td>
<td>short, int, long, float, or double</td>
</tr>
<tr>
<td>short</td>
<td>int, long, float, or double</td>
</tr>
<tr>
<td>char</td>
<td>int, long, float, or double</td>
</tr>
<tr>
<td>int</td>
<td>long, float, or double</td>
</tr>
<tr>
<td>long</td>
<td>float or double</td>
</tr>
<tr>
<td>float</td>
<td>double</td>
</tr>
</tbody>
</table>

How to use cast?

**Forcing floating point division between int expressions**

```
int qp = 35;
int credits = 10;
double gpa = (double) qp / credits;
```

gpa should be 3.5
### How to use cast?

*Scaling a double and converting to int*

```java
double gpa = 3.2;
int gpaPercent = (int) (gpa / 4) * 100;
```

- `gpaPercent` should be 80

```java
double gpa = 3.2;
int gpaPercent = (int) ((gpa / 4) * 100);
```

### Characters in Java

- A `char` variable stores a single character
- Character literals are delimited by single quotes:
  ```
  'a'   'X'   '7'   '$'   ','   '\n'
  ```

  Note the difference between a primitive character variable, which holds only one character, and a `String` object, which can hold multiple characters

```java
char grade = 'A';
char terminator = ';', separator = ' ', newline = '\n';
char letter = 't';
char next = letter++; // 'u'
char letterCode = letter; // 116 (code for 't')
int letterCode = letter; // 116 (code for 't')
char nextCode = next; // 117 (code for 'u')
int nextCode = (int) nextCode; // 117 (code for 'u')
char nextAlt = (char) nextCode; // 'u'

char whatsthis = (char) ('A' + 3); // ??
int num = (int) (letter - 'p'); // ??
```

### char ↔ int Conversion

- A `char` variable is stored as its unicode representation
- `char ↔ int` conversion: convert between char and its unicode.
- e.g.: `\t` ↔ 116
- Increment and decrement of `char` variables takes you up and down in alphabetical order (codes are in numeric sequence)

```java
char letter = 't';
int letterCode = letter; // 116 (code for 't')
char next = letter++; // 'u'
int nextCode = next; // 117 (code for 'u')
char nextAlt = (char) nextCode; // 'u'

char whatsthis = (char) ('A' + 3); // ??
int num = (int) (letter - 'p'); // ??
```