Characters and Strings

Lecture 13

CS106A, Summer 2019
Sarai Gould && Laura Cruz-Albrecht

With inspiration from slides created by Keith Schwarz, Mehran Sahami, Eric Roberts, Stuart Reges, Chris Piech, Brahm Capoor and others.
Announcements: Midterm

- **Midterm:** Monday July 22nd, 7-9PM in Bishop Auditorium
- Review session: Friday July 19th, 10:30AM in Gates B01

- If you have an academic or University conflict, email both instructors and fill out the following form **by tonight** 7/16
Announcements: Midterm

- Download BlueBook software on website
- Make sure you have two-factor authentication on Duo Mobile with **passcodes** set up; will need for submission
  - Two-Factor Authentication: Duo Mobile
  - How to Use the Duo Mobile Passcode for Two-Step Authentication
- Check out Website for other relevant Midterm information
Plan for Today

- Review: Memory
- Characters
- Strings
Plan for Today

- Review: Memory
- Characters
- Strings
Review: Memory

Primitives

- passed by value
- stored on the stack

Objects

- passed by reference
- stored on the heap, referred to from the stack
Review: Stack vs. Heap

The **Stack** is more temporary memory. Things on the Stack are added and removed as methods are called and returned.

The **Heap** is more permanent memory. Things on the Heap don’t disappear as methods are called or returned.
Review: Stack vs. Heap

```c
int x = 22;
```
Review: Stack vs. Heap

```c
int x = 22;
```

The Stack

```
x 22
```

The Heap
Review: Stack vs. Heap

\[
\text{int } x = 22; \\
\text{GRect rect1 = new GRect(30, 10);} \\
\]

The Stack

\[
\begin{array}{c}
\text{x} \\
22
\end{array}
\]

The Heap
Review: Stack vs. Heap

```java
int x = 22;
GRect rect1 = new GRect(30, 10);
```

The Stack

x 22

The Heap

4
int x = 22;
GRect rect1 = new GRect(30, 10);
Review: Stack vs. Heap

```java
int x = 22;
GRect rect1 = new GRect(30, 10);
GRect rect2 = new GRect(10, 10);
```
```
int x = 22;
GRect rect1 = new GRect(30, 10);
GRect rect2 = new GRect(10, 10);
```
Review: Stack vs. Heap

```java
int x = 22;
GRect rect1 = new GRect(30, 10);
GRect rect2 = new GRect(10, 10);
```

The Stack

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rect1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rect2</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
</tr>
</tbody>
</table>

The Heap

<table>
<thead>
<tr>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

== compares values in the Stack

.equals() compares objects on the Heap
public void run(){

    GRect rect1 = new GRect(100, 100); // rect1 = location 4
    GRect rect2 = new GRect(100, 100); // rect2 = location 5

    // is the memory address of rect1 equal to the memory address of rect2?
    if(rect1 == rect2){
        println("These rectangles are equal!");
    } else {
        // no! They’re at different locations in memory!
        println("Actually, these rectangles are not equal.");
    }
}
```java
public void run(){

    GRect rect3 = new GRect(100, 100);

    if(rect3 == rect3){
        // This is true!
        println("This rectangle is equal to itself!");
    } else {
        println("Actually, this rectangle is not equal to itself.");
    }
}
```

**Remember:** This will only evaluate to true if it's the exact same rectangle! This has something to do with how objects are stored in memory.
public void run(){
    GRect rect4 = new GRect(100, 100); // rect4 = location 10
    GRect rect5 = rect4; // rect5 also stores location 10
    // is the memory address of rect4 equal to the memory address of rect4?
    if(rect4 == rect5){
        // They are equal! They both point to the same place in memory!
        println("These rectangles are equal!");
    } else {
        println("Actually, these rectangles are not equal.");
    }
}
Pass by Reference vs Value

If something is passed by *reference*, it *can* be altered simply by passing it into a method. This is because we are passing in a *reference to its location in memory*, not a copy of the object.

If something is passed by *value*, it *cannot* be altered simply by passing it into a method. This is because we are a passing in a *copy of its value*.
Plan for Today

- Review: Memory
- Characters
- Strings
Text Processing
Text Processing

i like ice cream

Mi piace il gelato
Text Processing

i like ice cream  Mi piace il gelato
Text Processing

i like ice cream  Mi piace il gelato
Text Processing
Plan for Today

- Review: Memory
- Characters
- Strings
A **char** is a variable type that represents a single character or “glyph”.

```c
char letterA = 'A';
```
A `char` is a variable type that represents a single character or "glyph".

```plaintext
char letterA = 'A';
```
A `char` is a variable type that represents a single character or “glyph”.

```java
char letterA = 'A';
char plus = '+';
char zero = '0';
char space = ' ';```

A `char` is a variable type that represents a single character or “glyph”.

```java
char letterA = 'A';
char plus = '+';
char zero = '0';
char space = ' ';

// special characters
char newLine = '\n';
char tab = '\t';
char singleQuote = '\'';
char backSlash = '\\';
```
Char

Under the hood, Java represents each char as an integer. This integer is its “ASCII” value.
* This is only the first half of the table

** ASCII: American Standard Code for Information Interchange**
* This is only the first half of the table

```
char uppercaseA = 'A';    // Actually 65
```
char uppercaseA = 'A';  // Actually 65
char lowercaseA = 'a';  // Actually 97

* This is only the first half of the table
**ASCII**

<table>
<thead>
<tr>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>space</td>
<td>48</td>
<td>0</td>
<td>64</td>
<td>@</td>
<td>80</td>
<td>P</td>
<td>96</td>
<td>`</td>
<td>112</td>
<td>p</td>
<td>113</td>
<td>a</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>49</td>
<td>1</td>
<td>65</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
<td>a</td>
<td>113</td>
<td>q</td>
<td>114</td>
<td>b</td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>50</td>
<td>2</td>
<td>66</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
<td>b</td>
<td>114</td>
<td>r</td>
<td>115</td>
<td>c</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>51</td>
<td>3</td>
<td>67</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
<td>c</td>
<td>115</td>
<td>s</td>
<td>116</td>
<td>d</td>
</tr>
<tr>
<td>36</td>
<td>$</td>
<td>52</td>
<td>4</td>
<td>68</td>
<td>D</td>
<td>84</td>
<td>T</td>
<td>100</td>
<td>d</td>
<td>116</td>
<td>t</td>
<td>117</td>
<td>e</td>
</tr>
<tr>
<td>37</td>
<td>%</td>
<td>53</td>
<td>5</td>
<td>69</td>
<td>E</td>
<td>85</td>
<td>U</td>
<td>101</td>
<td>e</td>
<td>117</td>
<td>u</td>
<td>118</td>
<td>f</td>
</tr>
<tr>
<td>38</td>
<td>&amp;</td>
<td>54</td>
<td>6</td>
<td>70</td>
<td>F</td>
<td>86</td>
<td>V</td>
<td>102</td>
<td>f</td>
<td>118</td>
<td>c</td>
<td>119</td>
<td>g</td>
</tr>
<tr>
<td>39</td>
<td>,</td>
<td>55</td>
<td>7</td>
<td>71</td>
<td>G</td>
<td>87</td>
<td>W</td>
<td>103</td>
<td>g</td>
<td>119</td>
<td>w</td>
<td>120</td>
<td>h</td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>56</td>
<td>8</td>
<td>72</td>
<td>H</td>
<td>88</td>
<td>X</td>
<td>104</td>
<td>h</td>
<td>120</td>
<td>x</td>
<td>121</td>
<td>i</td>
</tr>
<tr>
<td>41</td>
<td>)</td>
<td>57</td>
<td>9</td>
<td>73</td>
<td>I</td>
<td>89</td>
<td>Y</td>
<td>105</td>
<td>i</td>
<td>121</td>
<td>y</td>
<td>122</td>
<td>j</td>
</tr>
<tr>
<td>42</td>
<td>+</td>
<td>58</td>
<td>:</td>
<td>74</td>
<td>J</td>
<td>90</td>
<td>Z</td>
<td>106</td>
<td>j</td>
<td>122</td>
<td>z</td>
<td>123</td>
<td>k</td>
</tr>
<tr>
<td>43</td>
<td>,</td>
<td>59</td>
<td>;</td>
<td>75</td>
<td>K</td>
<td>91</td>
<td>[</td>
<td>107</td>
<td>k</td>
<td>123</td>
<td>{</td>
<td>124</td>
<td>l</td>
</tr>
<tr>
<td>44</td>
<td>-</td>
<td>60</td>
<td>&lt;</td>
<td>76</td>
<td>L</td>
<td>92</td>
<td>\</td>
<td>108</td>
<td>l</td>
<td>124</td>
<td>`</td>
<td>125</td>
<td>m</td>
</tr>
<tr>
<td>45</td>
<td>&gt;</td>
<td>61</td>
<td>=</td>
<td>77</td>
<td>M</td>
<td>93</td>
<td>]</td>
<td>109</td>
<td>m</td>
<td>125</td>
<td>}</td>
<td>126</td>
<td>n</td>
</tr>
<tr>
<td>46</td>
<td>?</td>
<td>62</td>
<td>&gt;</td>
<td>78</td>
<td>N</td>
<td>94</td>
<td>^</td>
<td>110</td>
<td>n</td>
<td>126</td>
<td>~</td>
<td>127</td>
<td>o</td>
</tr>
<tr>
<td>47</td>
<td>/</td>
<td>63</td>
<td>?</td>
<td>79</td>
<td>O</td>
<td>95</td>
<td></td>
<td>111</td>
<td>o</td>
<td>127</td>
<td></td>
<td>128</td>
<td>p</td>
</tr>
</tbody>
</table>

* This is only the first half of the table

```cpp
char uppercaseA = 'A'; // Actually 65
char lowercaseA = 'a'; // Actually 97
char zeroDigit = '0'; // Actually 48
```
Other Useful ASCII Properties

<table>
<thead>
<tr>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>[space]</td>
<td>48</td>
<td>)</td>
<td>64</td>
<td>@</td>
<td>80</td>
<td>P</td>
<td>96</td>
<td>'</td>
<td>112</td>
<td>p</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>49</td>
<td>1</td>
<td>65</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
<td>a</td>
<td>113</td>
<td>q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>50</td>
<td>2</td>
<td>66</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
<td>b</td>
<td>114</td>
<td>r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>51</td>
<td>3</td>
<td>67</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
<td>c</td>
<td>115</td>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>$</td>
<td>52</td>
<td>4</td>
<td>68</td>
<td>D</td>
<td>84</td>
<td>T</td>
<td>100</td>
<td>d</td>
<td>116</td>
<td>t</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>%</td>
<td>53</td>
<td>5</td>
<td>69</td>
<td>E</td>
<td>85</td>
<td>U</td>
<td>101</td>
<td>e</td>
<td>117</td>
<td>u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>&amp;</td>
<td>54</td>
<td>6</td>
<td>70</td>
<td>F</td>
<td>86</td>
<td>V</td>
<td>102</td>
<td>f</td>
<td>118</td>
<td>v</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>'</td>
<td>55</td>
<td>7</td>
<td>71</td>
<td>G</td>
<td>87</td>
<td>W</td>
<td>103</td>
<td>g</td>
<td>119</td>
<td>w</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>56</td>
<td>8</td>
<td>72</td>
<td>H</td>
<td>88</td>
<td>X</td>
<td>104</td>
<td>h</td>
<td>120</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>)</td>
<td>57</td>
<td>9</td>
<td>73</td>
<td>I</td>
<td>89</td>
<td>Y</td>
<td>105</td>
<td>i</td>
<td>121</td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>'</td>
<td>58</td>
<td>:</td>
<td>74</td>
<td>J</td>
<td>90</td>
<td>Z</td>
<td>106</td>
<td>j</td>
<td>122</td>
<td>z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>+</td>
<td>59</td>
<td>;</td>
<td>75</td>
<td>K</td>
<td>91</td>
<td>[</td>
<td>107</td>
<td>k</td>
<td>123</td>
<td>{</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>,</td>
<td>60</td>
<td>&lt;</td>
<td>76</td>
<td>L</td>
<td>92</td>
<td>\</td>
<td>108</td>
<td>l</td>
<td>124</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>-</td>
<td>61</td>
<td>=</td>
<td>77</td>
<td>M</td>
<td>93</td>
<td>]</td>
<td>109</td>
<td>m</td>
<td>125</td>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>.</td>
<td>62</td>
<td>&gt;</td>
<td>78</td>
<td>N</td>
<td>94</td>
<td>^</td>
<td>110</td>
<td>n</td>
<td>126</td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>/</td>
<td>63</td>
<td>?</td>
<td>79</td>
<td>O</td>
<td>95</td>
<td>\</td>
<td>111</td>
<td>o</td>
<td>127</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Uppercase letters are sequential (‘A’ -> ‘Z’)
### Other Useful ASCII Properties

<table>
<thead>
<tr>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
<th>Code</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>space</td>
<td>48</td>
<td>@</td>
<td>80</td>
<td>P</td>
<td>96</td>
<td>`</td>
</tr>
<tr>
<td>33</td>
<td>`</td>
<td>49</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
<td>a</td>
</tr>
<tr>
<td>34</td>
<td>`</td>
<td>50</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
<td>b</td>
</tr>
<tr>
<td>35</td>
<td>`</td>
<td>51</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
<td>c</td>
</tr>
<tr>
<td>36</td>
<td>`</td>
<td>52</td>
<td>D</td>
<td>84</td>
<td>T</td>
<td>100</td>
<td>d</td>
</tr>
<tr>
<td>37</td>
<td>`</td>
<td>53</td>
<td>E</td>
<td>85</td>
<td>U</td>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>38</td>
<td>`</td>
<td>54</td>
<td>F</td>
<td>86</td>
<td>V</td>
<td>102</td>
<td>f</td>
</tr>
<tr>
<td>39</td>
<td>`</td>
<td>55</td>
<td>G</td>
<td>87</td>
<td>W</td>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>40</td>
<td>`</td>
<td>56</td>
<td>H</td>
<td>88</td>
<td>X</td>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>41</td>
<td>`</td>
<td>57</td>
<td>I</td>
<td>89</td>
<td>Y</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>42</td>
<td>`</td>
<td>58</td>
<td>J</td>
<td>90</td>
<td>Z</td>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>43</td>
<td>`</td>
<td>59</td>
<td>K</td>
<td>91</td>
<td>[</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>44</td>
<td>`</td>
<td>60</td>
<td>L</td>
<td>92</td>
<td>\</td>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>45</td>
<td>`</td>
<td>61</td>
<td>M</td>
<td>93</td>
<td>]</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>46</td>
<td>`</td>
<td>62</td>
<td>N</td>
<td>94</td>
<td>^</td>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>47</td>
<td>`</td>
<td>63</td>
<td>O</td>
<td>95</td>
<td>_</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td></td>
<td></td>
<td>79</td>
<td>EOF</td>
<td>112</td>
<td>`</td>
<td>113</td>
<td>p</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Uppercase letters are sequential ("A" -> "Z")
- Lowercase letters are sequential ("a" -> "z")
Other Useful ASCII Properties

- Uppercase letters are sequential (‘A’ -> ‘Z’)
- Lowercase letters are sequential (‘a’ -> ‘z’)
-Digits are sequential (‘0’ -> ‘9’)

![ASCII Table](image)
Char Math

We can take advantage of Java representing each char as an integer!
boolean areEqual = 'A' == 'A';    // true
boolean areEqual = 'A' == 'A'; // true
boolean earlierLetter = 'f' < 'c'; // false
```java
boolean areEqual = 'A' == 'A'; // true
boolean earlierLetter = 'f' < 'c'; // false
char uppercaseB = 'A' + 1; // 'B'
```
boolean areEqual = 'A' == 'A'; // true
boolean earlierLetter = 'f' < 'c'; // false
char uppercaseB = 'A' + 1; // 'B'
int diff = 'c' - 'a'; // 2
boolean areEqual = 'A' == 'A';   // true
boolean earlierLetter = 'f' < 'c'; // false
char uppercaseB = 'A' + 1;        // 'B'
int diff = 'c' - 'a';            // 2

int alphabetSize = 'z' - 'a' + 1;
// or
int alphabetSize = 'Z' - 'A' + 1;
// prints the numbers 1 to 20
for (int i = 1; i <= 20; i++) {
    println(i);
}
// prints the characters a to z
for (char ch = 'a'; ch <= 'z'; ch++) {
    println(ch);
}
Char Math

Not every integer maps to a character. So when you have an expression with ints and chars, Java picks int as the most expressive type.
Char Math

Not every integer maps to a character. So when you have an expression with ints and chars, Java picks int as the most expressive type.

Recall the expressive hierarchy:

String > double > int > char > boolean
Char Math

Not every integer maps to a character. So when you have an expression with ints and chars, Java picks int as the most expressive type.

'A' + 1
Char Math

Not every integer maps to a character. So when you have an expression with ints and chars, Java picks int as the most expressive type.

'A' + 1

char, with ASCII value 65

int
Char Math

Not every integer maps to a character. So when you have an expression with ints and chars, Java picks int as the most expressive type.

'A' + 1 // evaluates to 66 (int)

char, with ASCII value 65

int

Answer will be an int
Char Math

Not every integer maps to a character. So when you have an expression with ints and chars, Java picks int as the most expressive type.

'A' + 1 // evaluates to 66 (int)
'c' + (2 * 5) - 1 // evaluates to 108
Char Math

Not every integer maps to a character. So when you have an expression with ints and chars, Java picks int as the most expressive type.

'A' + 1 // evaluates to 66 (int)
'c' + (2 * 5) - 1 // evaluates to 108

We can make it a char by putting it in a char variable.
Char Math

Not every integer maps to a character. So when you have an expression with ints and chars, Java picks int as the most expressive type.

'\text{A}' + 1 \quad \text{// evaluates to 66 (int)}
'\text{c}' + (2 \times 5) - 1 \quad \text{// evaluates to 108}

We can make it a char by putting it in a char variable.

\text{char } \text{uppercaseB} = '\text{A}' + 1;
\text{char } \text{alsoUppercaseB} = 66;
If we want to force Java to treat an expression as a particular type, we can also cast it to that type.
Type-Casting

If we want to force Java to treat an expression as a particular type, we can also cast it to that type.

'A' + 1 // evaluates to 66 (int)
Type-Casting

If we want to force Java to treat an expression as a particular type, we can also cast it to that type.

'\texttt{A}' + 1 \quad // \text{evaluates to 66 (int)}
\texttt{(char)('A' + 1)} \quad // \text{evaluates to 'B' (char)}
Type-Casting

If we want to force Java to treat an expression as a particular type, we can also cast it to that type.

'A' + 1       // evaluates to 66 (int)
(char)('A' + 1) // evaluates to 'B' (char)
(char)'A' + 1   // evaluates to 66 (int)
Type-Casting

If we want to force Java to treat an expression as a particular type, we can also cast it to that type.

'\texttt{A} + 1 \quad // \text{evaluates to 66 (int)}$

\texttt{char}(\texttt{A} + 1) \quad // \text{evaluates to 'B' (char)}

\texttt{char}'\texttt{A}' + 1 \quad // \text{evaluates to 66 (int)}

\texttt{1 / 2} \quad // \text{evaluates to 0 (int)}
Type-Casting

If we want to force Java to treat an expression as a particular type, we can also cast it to that type.

'A' + 1 // evaluates to 66 (int)
(char)('A' + 1) // evaluates to 'B' (char)
(char)'A' + 1 // evaluates to 66 (int)

1 / 2 // evaluates to 0 (int)
(double)1 / 2 // evaluates to 0.5 (double)
Type-Casting

If we want to force Java to treat an expression as a particular type, we can also cast it to that type.

'A' + 1    // evaluates to 66 (int)
(char)('A' + 1)    // evaluates to 'B' (char)
(char)'A' + 1    // evaluates to 66 (int)

1 / 2    // evaluates to 0 (int)
(double)1 / 2    // evaluates to 0.5 (double)
1 / (double)2    // evaluates to 0.5 (double)
Type-Casting

If we want to force Java to treat an expression as a particular type, we can also cast it to that type.

'A' + 1 // evaluates to 66 (int)
(char)'A' + 1 // evaluates to 'B' (char)
(char)'A' + 1 // evaluates to 66 (int)

1 / 2 // evaluates to 0 (int)
(double)1 / 2 // evaluates to 0.5 (double)
1 / (double)2 // evaluates to 0.5 (double)
(double)(1 / 2) // eek! evaluates to 0.0 (double)
There are several helpful built-in Java methods to manipulate chars.
There are several helpful built-in Java methods to manipulate chars.

```java
char lowercaseA = 'a';
char uppercaseA = Character.toUpperCase(lowercaseA);
```
There are several helpful built-in Java methods to manipulate chars.

```java
char lowercaseA = 'a';
char uppercaseA = Character.toUpperCase(lowercaseA);

char plus = '+';
if (Character.isLetter(plus)) {
    ... // Does not execute: + is not a letter
}
```
### Character Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>boolean Character.isDigit(char ch)</code></td>
<td>Determines if the specified character is a digit.</td>
</tr>
<tr>
<td><code>boolean Character.isLetter(char ch)</code></td>
<td>Determines if the specified character is a letter.</td>
</tr>
<tr>
<td><code>boolean Character.isLetterOrDigit(char ch)</code></td>
<td>Determines if the specified character is a letter or a digit.</td>
</tr>
<tr>
<td><code>boolean Character.isLowerCase(char ch)</code></td>
<td>Determines if the specified character is a lowercase letter.</td>
</tr>
<tr>
<td><code>boolean Character.isUpperCase(char ch)</code></td>
<td>Determines if the specified character is an uppercase letter.</td>
</tr>
<tr>
<td><code>boolean Character.isWhitespace(char ch)</code></td>
<td>Determines if the specified character is whitespace (spaces and tabs).</td>
</tr>
<tr>
<td><code>char Character.toLowerCase(char ch)</code></td>
<td>Converts <code>ch</code> to its lowercase equivalent, if any. If not, <code>ch</code> is returned unchanged.</td>
</tr>
<tr>
<td><code>char Character.toUpperCase(char ch)</code></td>
<td>Converts <code>ch</code> to its uppercase equivalent, if any. If not, <code>ch</code> is returned unchanged.</td>
</tr>
</tbody>
</table>
Character Methods

toLowerCase and toUpperCase return the new char; they cannot modify an existing char!

Always save the return value of Character methods!
Character Methods

toLowerCase and toUpperCase return the new char; they cannot modify an existing char!

Always save the return value of Character methods!

```java
char letter = 'a';
Character.toUpperCase(letter); // Does nothing!
println(letter); // prints 'a'
```
Character Methods

toLowerCase and toUpperCase return the new char; they cannot modify an existing char!

Always save the return value of Character methods!

```java
char letter = 'a';
Character.toUpperCase(letter); // Does nothing!
println(letter); // prints 'a'

char letter = 'a';
char newLetter = Character.toUpperCase(letter);
println(newLetter); // prints 'A' :)
```
Plan for Today

- Review: Memory
- Characters
- Strings
Strings
String

Text is stored using the variable type String.
A String is a sequence of characters!

String text = “Hello!”;
String

Text is stored using the variable type String. A String is a sequence of characters!

```
String text = "Hello!";
```
Hello!
Hello!
public void run() {
    String text = "hello!";
}

How it’s actually stored

Stack

run

Heap

Length: 6

Hello!
The `string.length()` method returns the number of characters in the string. This is one larger than the last valid index in the string.

```
Hello!
```

```
int strLen = text.length();  // 6
```
String Methods: charAt

The `string.charAt(index)` method returns the character at a given index.

```java
char first = text.charAt(0); // 'H'
```

strLen: 6
String Methods: charAt

The `string.charAt(index)` method returns the character at a given index. The index must be between 0 and `length - 1`.

```
Hello!
```

```
char first = text.charAt(0);  // 'H'
char last = text.charAt(strLen - 1);  // '!' 
```
**String Methods: charAt**

The `string.charAt(index)` method returns the character at a given index.

```java
char first = text.charAt(0); // 'H'
char last = text.charAt(strLen - 1); // '!' 
char bad = text.charAt(strLen); // error
```
Exercise

String $q =$

“Can you think of my favorite hobby?”
Exercise

String $q =$

“Can you think of my favorite hobby?”
Exercise

String \( q = \)

“Can you think of my favorite hobby?”

“” + \( q.\text{charAt}(12) \) + \( q.\text{charAt}(2) \) + \( q.\text{charAt}(10) \) + \( q.\text{charAt}(26) \)
String q =

“Can you think of my favorite hobby?”

“” + q.charAt(12) + q.charAt(2) + q.charAt(10) + q.charAt(26)

knit
Exercise

String q =

“Can you think of my favorite hobby?”

“” + q.charAt(12) + q.charAt(2) + q.charAt(10) + q.charAt(26)

knit

* fun fact: also uses Strings
Substrings

A substring is a subset of a string.

String \texttt{str} = “Hi Duke!”;
String \texttt{hi} = \texttt{str}.substring(0, 2);
Substrings

A **substring** is a subset of a string.

String `str = “Hi Duke!”;
String `dukeExclm = str.substring(3);  // to end
## Useful String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int length()</td>
<td>Returns the length of the string</td>
</tr>
<tr>
<td>char charAt(int index)</td>
<td>Returns the character at the specified index. Note: Strings indexed starting at 0.</td>
</tr>
<tr>
<td>String substring(int p1, int p2)</td>
<td>Returns the substring beginning at p1 and extending up to but not including p2</td>
</tr>
<tr>
<td>String substring(int p1)</td>
<td>Returns substring beginning at p1 and extending through end of string.</td>
</tr>
<tr>
<td>boolean equals(String s2)</td>
<td>Returns true if string s2 is equal to the receiver string. This is case sensitive.</td>
</tr>
<tr>
<td>int compareTo(String s2)</td>
<td>Returns integer whose sign indicates how strings compare in lexicographic order</td>
</tr>
<tr>
<td>int indexOf(char ch) or int indexOf(String s)</td>
<td>Returns index of first occurrence of the character or the string, or -1 if not found</td>
</tr>
<tr>
<td>String toLowerCase() or String toUpperCase()</td>
<td>Returns a lowercase or uppercase version of the receiver string</td>
</tr>
</tbody>
</table>

*remember, called using dot notation: `myString.length()`*
String str = "Hello, world!";
Creating Strings

String str = "Hello, world!";
String empty = "";
String \texttt{str} = "Hello, world!";
String \texttt{empty} = "";

// Read in text from the user
String \texttt{name} = \texttt{readLine("What is your name? ");}
Creating Strings

String str = "Hello, world!";
String empty = "";

// Read in text from the user
String name = readLine("What is your name? ");

// String concatenation (using "+")
String message = 2 + " be or not " + 2 + " be";
Creating Strings

String str = "Hello, world!";
String empty = "";

// Read in text from the user
String name = readLine("What is your name? ");

// String concatenation (using “+”)
String message = 2 + " be or not " + 2 + " be";

int x = 2;
println("x has the value " + x);
Strings are Immutable

Java strings are **immutable**: once you create a String, its contents cannot be changed.

```java
// Cannot change individual chars in the string
String typo = "Hello, warld!";
typo.charAt(8) = 'o'; // Error! Will not run.
```
Strings are Immutable

Java strings are **immutable**: once you create a String, its contents cannot be changed.

```java
// Cannot change individual chars in the string
String typo = "Hello, warld!";
typo.charAt(8) = 'o'; // Error! Will not run.

To change a String, you must create a new String containing the value you want (e.g. using String methods).

String corrected = "Hello, world!";
```
Strings are Immutable

Important consequence: if you pass a String into a method, that method cannot modify that string.

```
String className = "cs 106a";
className.toUpperCase(); // does nothing!

className = className.toUpperCase();
println(className); // CS 106A
```
public void run(){

    String str1 = “Hello!”;
    String str2 = “Hello!”;

    if(str1 == str2){
        println(“These Strings are equal!”);
    } else{
        println(“Actually, these Strings are not equal.”);
    }
}
public void run() {

    String str1 = "Hello!";
    String str2 = "Hello!";

    if (str1 == str2) {
        println("These Strings are equal!");
    } else {
        println("Actually, these Strings are not equal.");
    }
}
public void run(){

    String str1 = “Hello!”;
    String str2 = “Hello!”;

    if(str1 == str2){
        println(“These Strings are equal!”);
    } else {
        println(“Actually, these Strings are not equal.”);
    } 
}
public void run(){

    String str1 = "Hello!";
    String str2 = "Hello!";

    if(str1 == str2){
        println("These Strings are equal!");
    } else {
        println("Actually, these Strings are not equal.");
    }
}
public void run(){

    String str1 = "Hello!";
    String str2 = "Hello!";

    if(str1 == str2){
        println("These Strings are equal!");
    } else {
        println("Actually, these Strings are not equal.");
    }
}

run
Hello!
\begin{tabular}{|c|}
\hline
28
\hline
\end{tabular}
\begin{tabular}{|c|}
\hline
str1
\hline
\end{tabular}
\begin{tabular}{|c|}
\hline
28
\hline
\end{tabular}
public void run(){
    String str1 = "Hello!";
    String str2 = "Hello!";
    if(str1 == str2){
        println("These Strings are equal!");
    } else {
        println("Actually, these Strings are not equal.");
    }
}
public void run(){

    String str1 = "Hello!";
    String str2 = "Hello!";

    if(str1 == str2){
        println("These Strings are equal!");
    } else {
        println("Actually, these Strings are not equal.");
    }
}
public void run(){

    String str1 = “Hello!”;
    String str2 = “Hello!”;

    if(str1 == str2){
        println(“These Strings are equal!”);
    } else {
        println(“Actually, these Strings are not equal.”);
    }
}
public void run(){
    String str1 = “Hello!”;
    String str2 = “Hello!”;
    if(str1 == str2){
        println(“These Strings are equal!”);
    } else {
        println(“Actually, these Strings are not equal.”);
    }
}
public void run(){
    String str1 = “Hello!”;
    String str2 = “Hello!”;

    if(str1 == str2){
        println(“These Strings are equal!”);
    } else {
        println(“Actually, these Strings are not equal.”);
    }
}
Instead Use...

```java
str1.equals(str2)
```
public void run(){

    String str1 = "Hello!";
    String str2 = "Hello!";

    if (str1.equals(str2)){
        println("These Strings have the same text!");
    } else {
        println("These Strings do NOT have the same text.");
    }
}
## Comparing Strings

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s1.equals(s2)</code></td>
<td>whether two strings contain the same characters</td>
</tr>
<tr>
<td><code>s1.equalsIgnoreCase(s2)</code></td>
<td>whether two strings contain the same characters, ignoring upper vs. lower case</td>
</tr>
<tr>
<td><code>s1.startsWith(s2)</code></td>
<td>whether <code>s1</code> contains <code>s2</code>’s characters at start</td>
</tr>
<tr>
<td><code>s1.endsWith(s2)</code></td>
<td>whether <code>s1</code> contains <code>s2</code>’s characters at end</td>
</tr>
<tr>
<td><code>s1.contains(s2)</code></td>
<td>whether <code>s2</code> is found within <code>s1</code></td>
</tr>
</tbody>
</table>
A common String programming pattern is looping over a String and operating on each character.

```java
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    // do something
}
```
// Creates a new String in all caps
String str = "Hello!";
String newStr = "";

for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    newStr += Character.toUpperCase(ch);
}

println(newStr);  // HELLO
Given two strings, how can we check if one starts with the other?
/*
 * startsWith
 * ----------
 * This method returns whether the String s1 starts with
 * the String s2. Can assume s1 is as long as or longer than s2.
 */

private boolean startsWith(String s1, String s2) {
    for (int i = 0; i < s2.length(); i++) {
        if (s1.charAt(i) != s2.charAt(i)) {
            return false;
        }
    }
    return true;
}
Plan for Today

- Review: Memory
- Characters
- Strings

Reminder: Check out Midterm Info page on website
  → Download Bluebook
  → Set up Two-Factor with Passcodes

Next time: Problem Solving with Strings!