Random numbers
The Random class

- Objects of the Random class generate pseudo-random numbers.
  - Class Random is found in the java.util package.

```java
import java.util.*;
```

- The methods of a Random object

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextInt()</td>
<td>returns a random integer</td>
</tr>
<tr>
<td>nextInt(max)</td>
<td>returns a random integer in the range [0, max)</td>
</tr>
<tr>
<td></td>
<td>in other words, from 0 to one less than max</td>
</tr>
<tr>
<td>nextDouble()</td>
<td>returns a random real number in the range [0.0, 1.0)</td>
</tr>
</tbody>
</table>
Generating random numbers

Random rand = new Random();
int randomNum = rand.nextInt(10);
// randomNum has a random value between 0 and 9

- What if we wanted a number from 1 to 10?
  int randomNum = rand.nextInt(10) + 1;

- What if we wanted a number from min to max (i.e. an arbitrary range)?
  int randomNum = rand.nextInt(<size of the range>) + <min>

  where <size of the range> equals (<max> - <min> + 1)
Random questions

- Given the following declaration, how would you get:
  ```java
  Random rand = new Random();
  ```
  - A random number between 0 and 100 inclusive?
  - A random number between 1 and 100 inclusive?
  - A random number between 4 and 17 inclusive?
Random solutions

Given the following declaration, how would you get:
```
Random rand = new Random();
```

- A random number between 0 and 100 inclusive?
  ```java
  int random1 = rand.nextInt(101);
  ```

- A random number between 1 and 100 inclusive?
  ```java
  int random1 = rand.nextInt(100) + 1;
  ```

- A random number between 4 and 17 inclusive?
  ```java
  int random1 = rand.nextInt(14) + 4;
  ```
Practice

- How might we:
  - Simulate a coin toss?
  - Rolling two dice?
Text processing
Text processing

- **text processing**: Examining, editing, formatting text.
  - Text processing often involves **for** loops that examine the characters of a string one by one.

- **Two data types involved**

<table>
<thead>
<tr>
<th>char</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Represents <em>individual characters</em></td>
<td>Represents <strong>sequences of characters</strong></td>
</tr>
<tr>
<td>Primitive type</td>
<td>Object type (i.e., not primitive)</td>
</tr>
<tr>
<td>Written with single quotes</td>
<td>Written with double quotes</td>
</tr>
<tr>
<td>e.g.: ‘T’</td>
<td>e.g.: “We the people”</td>
</tr>
</tbody>
</table>
| ‘t’, ‘3’, ‘%’ | “1. Twas brillig, and the slithy toves
| ‘\n’ | “” |

<table>
<thead>
<tr>
<th>char</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘T’</td>
<td>“1. Twas brillig, and the slithy toves”</td>
</tr>
<tr>
<td>‘\n’</td>
<td>“”</td>
</tr>
</tbody>
</table>

"T"
Characters

- **char**: A *primitive* type representing single characters.

- Individual characters inside a *String* are stored as char values.

- Literal char values are surrounded with apostrophe (single-quote) marks, such as 'a' or '4' or '\n' or '\'

- Like any other type, you can create variables, parameters, and returns of type char.

  ```java
  char letter = 'S';
  System.out.println(letter);    // prints S
  ```
Strings

- **String**: an object type for representing *sequences* of characters

  - Sequence can be of length 0, 1, or longer
  - Each element of the sequence is a char
  - We write strings surrounded in double-quotes
  - We can declare, initialize, assign, and use String variables in expressions just like other data types

```java
String s = "Hello, world\n";       // declare, init
System.out.println(s);             // use value
s = s + "I am your master\n";      // concatenate
                                 // and assign
```
Unlike primitive types, **object types can have methods**. Here is a list of methods for strings:

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charAt(index)</td>
<td>returns the character at the given index</td>
</tr>
<tr>
<td>indexOf(str)</td>
<td>returns the index where the start of the given string appears in this string (-1 if not found)</td>
</tr>
<tr>
<td>length()</td>
<td>returns the number of characters in this string</td>
</tr>
<tr>
<td>substring(index1, index2)</td>
<td>returns the characters in this string from index1 up to, but not including, index2</td>
</tr>
<tr>
<td>toLowerCase()</td>
<td>returns a new string with all lowercase letters</td>
</tr>
<tr>
<td>toUpperCase()</td>
<td>returns a new string with all uppercase letters</td>
</tr>
</tbody>
</table>
The `charAt` method

- The characters of a string can be accessed using the `String` object's `charAt` method.
  - String indices start at 0.

```java
String word = "cola";
char firstLetter = word.charAt(0);
if (firstLetter == 'c') {
    System.out.println("C is for cookie!");
}
```
Calling string methods

- Let $s$ be a variable of type String
- General syntax for calling a String method:
  
  ```java
  s.<method>(<args>)
  ```

- Some examples:
  String $s$ = “Cola”;
  int len = $s$.length();  // len == 4
  char firstLetter = $s$.charAt(0);  // ‘C’
  int index = $s$.indexOf(“ol”);  // index == 1
  String sub = $s$.substring(1,3);  // “ol”
  String up = $s$.toUpperCase();  // “COLA”
  String down = $s$.toLowerCase();  // “cola”
Fun with char!

- **char values can be concatenated with strings.**
  ```java
  char initial = 'P';
  System.out.println(initial + " . Diddy");
  ```

- **You can compare char values with relational operators.**
  - 'a' < 'b' and 'Q' != 'q'
  - *Caution:* You should **NOT** use these operators on a String!
char vs. String

- 'h' is a char
  char c = 'h';
  - char values are **primitive**; you cannot call methods on them
  - can't say c.length() or c.toUpperCase()

- "h" is a String
  String s = "h";
  - Strings are objects; they contain methods that can be called
  - can say s.length() 1
  - can say s.toUpperCase() "H"
  - can say s.charAt(0) 'h'
Numbers vs Strings

- 345 is an int
  
  ```java
  int i = 345;
  ```
  - int values are **primitive**; you cannot call methods on them
  - CAN perform arithmetic: `i = i * 2; // i==690`
  - CANNOT say `i.length()` or `i.charAt(0)`

- "345" is a String
  
  ```java
  String s = "345";
  ```
  - Strings are objects; they contain methods that can be called
  - can say `s.length()`  // returns 3
  - can say `s.charAt(1)`  // returns ‘4’
  - CANNOT perform arithmetic: `s = s * 2; // ERROR!`
More text processing: Comparing strings

- Objects (such as String) should be compared for equality by calling a method named equals.

- Example:
  ```java
  Scanner console = new Scanner(System.in);
  System.out.print("What is your name? ");
  String name = console.next();
  if (name.equals("Barney")) {
      System.out.println("I love you, you love me,");
      System.out.println("We're a happy family!");
  }
  ```
More text processing: Comparing strings

- There are more methods of a String object that can be used in `<test>` conditions.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>equals(str)</code></td>
<td>whether this string contains exactly the same characters as the other string</td>
</tr>
<tr>
<td><code>equalsIgnoreCase(str)</code></td>
<td>whether this string contains the same characters as the other, ignoring upper-</td>
</tr>
<tr>
<td></td>
<td>vs. lowercase differences</td>
</tr>
<tr>
<td><code>startsWith(str)</code></td>
<td>whether this string’s beginning matches the argument</td>
</tr>
<tr>
<td><code>endsWith(str)</code></td>
<td>whether this string’s end matches the argument</td>
</tr>
</tbody>
</table>
Comparing strings: Examples

- Hypothetical examples, assuming the existence of various String variables:

  ```java
  if (title.endsWith("M.D.")) {
      System.out.println("What's your number?");
  }

  if (fullName.startsWith("Giorgio")) {
      System.out.println("When are you retiring?");
  }

  if (lastName.equalsIgnoreCase("lumBerg")) {
      System.out.println("I need your TPS reports!");
  }

  if (name.toLowerCase().indexOf("sr.") >= 0) {
      System.out.println("You must be old!");
  }
  ```
Data type conversion

Sometimes, we want to change between types.

For example, 2640 feet = x in miles?

```java
int x = 2640;
int y = 5280;

return x / y; // integer arithmetic
```

This returns 0 if 3000 and 5280 are both ints.
Data type conversion

Sometimes, we want to change between types.

For example, 3000 feet = x in miles?

```java
int x = 2640;
int y = 5280;
double z = (double) x;
return z / y;  // double arithmetic
```
Typecasting

An expression like this:

\((<\text{type}>) \ <\text{expression}>\)

is called a \emph{typecast}. It converts the expression into an equivalent expression of type \textless type\textgreater{}.

\textbf{Examples:}

\begin{align*}
(\text{double}) \ 2640 & \quad == \quad 2640.0 \\
(\text{int}) \ 32.7 & \quad == \quad 32 \\
(\text{int}) \ 'A' & \quad == \quad ?? \\
(\text{char}) \ 97 & \quad == \quad ?? \\
(\text{String}) \ 'A' & \quad == \quad ?? \\
(\text{String}) \ 27 & \quad == \quad ??
\end{align*}
Typecasting

An expression like this:

(<type>) <expression>

is called a typecast. It converts the expression into an equivalent expression of type <type>.

Examples:

(double) 2640  == 2640.0
(int) 32.7    == 32
(int) ‘A’     == 65
(char) 97     == ‘a’
(String) ‘A’  == TypeError
(String) 27    == TypeError
What about String typecasts?

The \texttt{(String)} typecasts don’t work with primitive types, like \texttt{char}, \texttt{int}, \texttt{and} \texttt{double}.

However, there are easy ways to convert to and from strings:

**Converting to strings:**

\begin{itemize}
  \item \texttt{int x to String:} \texttt{x + "" or "" + x}
  \item \texttt{double x to String:} \texttt{x + "" or "" + x}
  \item \texttt{char, boolean, or float x to String:} \texttt{same thing}
\end{itemize}
What about String typecasts?

The (String) typecasts don’t work with primitive types, like char, int, and double.

However, there are easy ways to convert to and from strings:

**Converting from strings:**
- String s to int: `Integer.parseInt(x)`
- String s to double: `Double.parseDouble(x)` ...
- String s to char: `Character.parseChar(x)`
Reading in Strings

Remember the Scanner method called `nextInt()`?

You can also use Scanners to read in Strings with the `next()` and `hasNext()` methods.

Example:

Please enter your name: Alexander Pieter Yates
Name 1 has 9 letters
Name 2 has 6 letters
Name 3 has 5 letters