Java's `Math` class

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.ceil(`*value*`)`	rounds up
`Math.floor(`*value*`)`	rounds down
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1*`,` *value2*`)`	larger of two values
`Math.min(`*value1*`,` *value2*`)`	smaller of two values
`Math.pow(`*base*`,` *exp*`)`	*base* to the *exp* power
`Math.random()`	random `double` between 0 and 1
`Math.round(`*value*`)`	nearest whole number
`Math.sqrt(`*value*`)`	square root

`Math.sin(`*value*`)`
`Math.cos(`*value*`)`
`Math.tan(`*value*`)`
	sine/cosine/tangent of an angle in radians

`Math.toDegrees(`*value*`)`
`Math.toRadians(`*value*`)`
	convert degrees to radians and back

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

Calling `Math` methods

`Math.`**methodName**`(`**parameters**`)`

- Examples:

```
double squareRoot = Math.sqrt(121.0);
System.out.println(squareRoot);              // 11.0

int absoluteValue = Math.abs(-50);
System.out.println(absoluteValue);           // 50

System.out.println(Math.min(3, 7) + 2);      // 5
```

- The `Math` methods do not print to the console.
  - Each method produces ("returns") a numeric result.
  - The results are used as expressions (printed, stored, etc.).

Strings

- A ***string*** is a sequence of characters.

- A ***string literal*** surrounds a character sequence with double quotes, as in "Hello", "52 Main St.", or "42", vs. an integer literal like 42 or character literal like 'a'.

Strings

- **string**: An object storing a sequence of text characters.

```
String name = "text";
String name = expression;
```

  - Examples:

```
String name = "Marla Singer";

int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

Indexes

- Characters of a string are numbered with 0-based *indexes*:

```
String name = "R. Kelly";
```

index	0	1	2	3	4	5	6	7
character	`R`	`.`		`K`	`e`	`l`	`l`	`y`

  - First character's index : 0
  - Last character's index : 1 less than the string's length

  - The individual characters are values of type `char` (seen later)

`String` methods

Method name	Description
`indexOf(`**`str`**`)`	index where the start of the given string appears in this string (-1 if not found)
`length()`	number of characters in this string
`substring(`**`index1`**`, `**`index2`**`)` or `substring(`**`index1`**`)`	the characters in this string from *index1* (inclusive) to *index2* (exclusive); if *index2* is omitted, grabs till end of string
`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String gangsta = "Dr. Dre";
System.out.println(gangsta.length());    // 7
```

String method examples

```
// index      012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";

System.out.println(s1.length());          // 12
System.out.println(s1.indexOf("e"));      // 8
System.out.println(s1.substring(7, 10)); // "Reg"

String s3 = s2.substring(1, 7);
System.out.println(s3.toLowerCase());     // "arty s"
```

- Given the following string:

```
// index          0123456789012345678901
String book = "Building Java Programs";
```

  – How would you extract the word `"Java"` ?

Modifying strings

- Methods like `substring` and `toLowerCase` build and return a new string, rather than modifying the current string.

```
String s = "lil bow wow";
s.toUpperCase();
System.out.println(s);    // lil bow wow
```

- To modify a variable's value, you must reassign it:

```
String s = "lil bow wow";
s = s.toUpperCase();
System.out.println(s);    // LIL BOW WOW
```

Interactive Programs with `Scanner`

Input and `System.in`

- **interactive program**: Reads input from the console.
  - While the program runs, it asks the user to type input.
  - The input typed by the user is stored in variables in the code.

  - Can be tricky; users are unpredictable and misbehave.
  - But interactive programs have more interesting behavior.

- **`Scanner`**: An object that can read input from many sources.
  - Communicates with `System.in` (the opposite of `System.out`)
  - Can also read from files , web sites, databases, ...

`Scanner` syntax

- The `Scanner` class is found in the `java.util` package.

```
import java.util.*;    // so you can use Scanner
```

- Constructing a `Scanner` object to read console input:

```
Scanner name = new Scanner(System.in);
```

  - Example:

```
Scanner console = new Scanner(System.in);
```

`Scanner` methods

Method	Description
`nextInt()`	reads an `int` from the user and returns it
`nextDouble()`	reads a `double` from the user
`next()`	reads a one-word `String` from the user
`nextLine()`	reads a one-*line* `String` from the user

- Each method waits until the user presses Enter.
- The value typed by the user is returned.

```
System.out.print("How old are you? ");  // prompt
int age = console.nextInt();
System.out.println("You typed " + age);
```

- **prompt**: A message telling the user what input to type.

Scanner example

```
import java.util.*;    // so that I can use Scanner

public class UserInputExample {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("How old are you? ");
        int age = console.nextInt();

        int years = 65 - age;
        System.out.println(years + " years to retirement!");
    }
}
```

```
age    29
years  36
```

- Console (user input underlined):

```
How old are you? 29
36 years until retirement!
```

Scanner example 2

```
import java.util.*;    // so that I can use Scanner

public class ScannerMultiply {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Please type two numbers: ");
        int num1 = console.nextInt();
        int num2 = console.nextInt();

        int product = num1 * num2;
        System.out.println("The product is " + product);
    }
}
```

- Output (user input underlined):

```
Please type two numbers: 8 6
The product is 48
```

  - The `Scanner` can read multiple values from one line.

Input tokens

- **token**: A unit of user input, as read by the `Scanner`.
  - Tokens are separated by *whitespace* (spaces, tabs, new lines).

- When a token is not the type you ask for, it crashes.

```
System.out.print("What is your age? ");
int age = console.nextInt();
```

Output:

```
What is your age? Timmy
java.util.InputMismatchException
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        ...
```

Strings as user input

- `Scanner`'s `next` method reads a word of input as a `String`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
name = name.toUpperCase();
System.out.println(name + " has " + name.length() +
    " letters and starts with " + name.substring(0, 1));
```

Output:

```
What is your name? Chamillionaire
CHAMILLIONAIRE has 14 letters and starts with C
```

- The `nextLine` method reads a line of input as a `String`.

```
System.out.print("What is your address? ");
String address = console.nextLine();
```

Characters

Type `char`

- **`char`** : A primitive type representing single characters.
  - A `String` is stored internally as an array of `char`

```
String s = "Ali G.";
```

index	*0*	*1*	*2*	*3*	*4*	*5*
value	`'A'`	`'l'`	`'i'`	`' '`	`'G'`	`'.'`

  - It is legal to have variables, parameters, returns of type `char`
    - surrounded with apostrophes: `'a'` or `'4'` or `'\n'` or `'\''`

```
char letter = 'P';
System.out.println(letter);              // P
System.out.println(letter + " Diddy");   // P Diddy
```

The `charAt` method

- The `char`s in a `String` can be accessed using the `charAt` method.
  - accepts an `int` index parameter and returns the `char` at that index

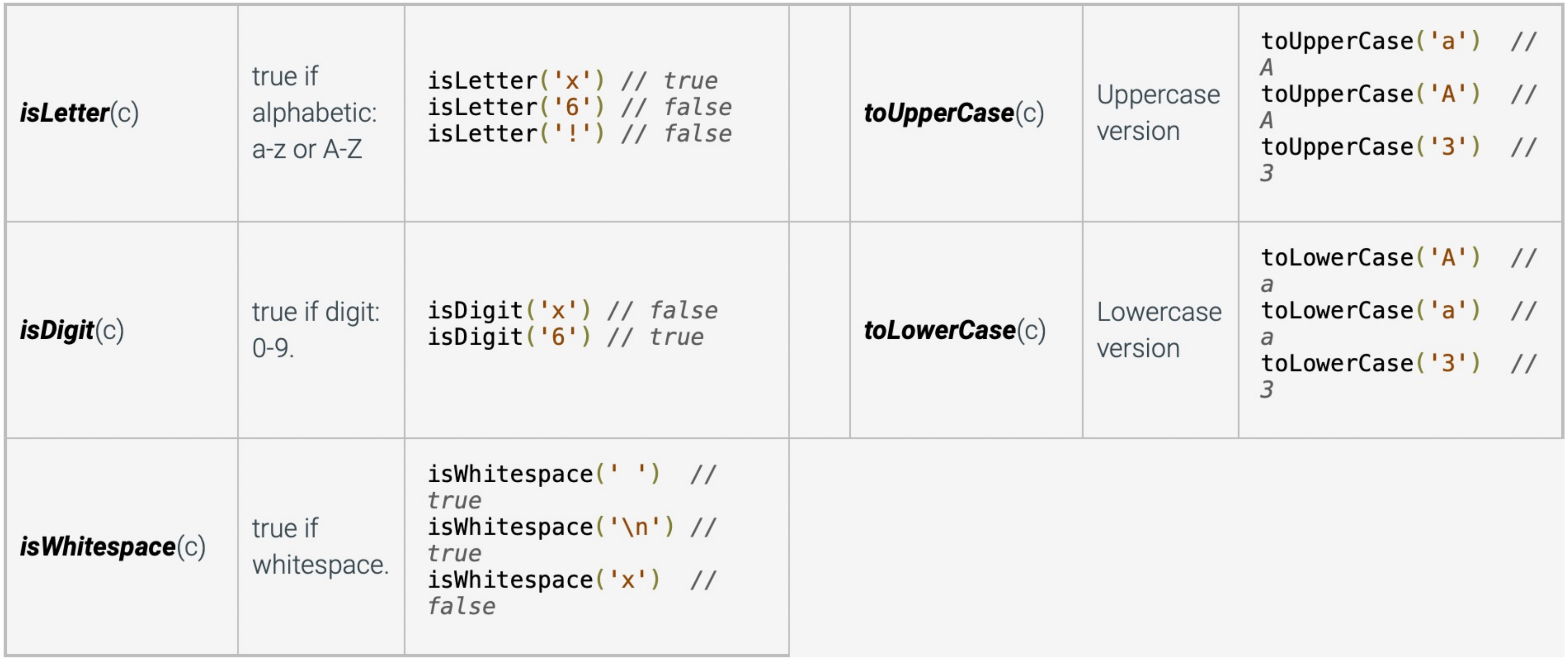
```
String food = "cookie";
char firstLetter = food.charAt(0);    // 'c'

System.out.println(firstLetter + " is for " + food);
```

- You can use a `for` loop to print or examine each character.

```
String major = "CSE";
for (int i = 0; i < major.length(); i++) {    // output:
    char c = major.charAt(i);                  // C
    System.out.println(c);                     // S
}                                              // E
```

Character operations

Table 3.15.1: Character methods return values. Each method must prepend Character., as in Character.isLetter.



isLetter(c)	true if alphabetic: a-z or A-Z	`isLetter('x') // true` `isLetter('6') // false` `isLetter('!') // false`		***toUpperCase***(c)	Uppercase version	`toUpperCase('a') // A` `toUpperCase('A') // A` `toUpperCase('3') // 3`
isDigit(c)	true if digit: 0-9.	`isDigit('x') // false` `isDigit('6') // true`		***toLowerCase***(c)	Lowercase version	`toLowerCase('A') // a` `toLowerCase('a') // a` `toLowerCase('3') // 3`
isWhitespace(c)	true if whitespace.	`isWhitespace(' ') // true` `isWhitespace('\n') // true` `isWhitespace('x') // false`				

Comparing `char` values

- You can compare `char`s with `==`, `!=`, and other operators:

```
String word = console.next();
char last = word.charAt(word.length() - 1);
if (last == 's') {
    System.out.println(word + " is plural.");
}

// prints the alphabet
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
}
```

`char` vs. `int`

- Each `char` is mapped to an integer value internally
- Called an **ASCII value** (You can find it at zybook 2.14)

`'A'`	is	65	`'B'`	is	66	`' '`	is	32
`'a'`	is	97	`'b'`	is	98	`'*'`	is	42

– Mixing `char` and `int` causes automatic conversion to `int`.

`'a' + 10` is 107, `'A' + 'A'` is 130

– To convert an `int` into the equivalent `char`, type-cast it.

`(char) ('a' + 2)` is `'c'`

char vs. String

- "h" is a String, but 'h' is a char (they are different)

- A String is an object; it contains methods.

```
String s = "h";
s = s.toUpperCase();        // "H"
int len = s.length();       //  1
char first = s.charAt(0);   // 'H'
```

- A char is primitive; you can't call methods on it.

```
char c = 'h';
c = c.toUpperCase();                // ERROR
s = s.charAt(0).toUpperCase();      // ERROR
```

  - What is `s + 1`? What is `c + 1`?
  - What is `s + s`? What is `c + c`?