Mathematical Functions, Characters, and Strings

CSE 114, Computer Science 1

Stony Brook University

http://www.cs.stonybrook.edu/~cse114

Static methods

- Remember the main method header?

```
public static void main(String[] args)
```

  - What does **static mean?**
    - associates a method with a particular class name
    - any method can call a **static method either:**
      - directly from within same class OR
      - using class name from outside class
  - Application Programming Interface (API) is the list of all public members of a class

The `Math` Class API

- Class constants (always static):
  - `PI`
  - `E`
- Class static methods:
  - Trigonometric methods
  - Exponent methods
  - Rounding methods
  - min, max, abs, and random methods

Trigonometric Methods

- `sin(double a)`
- `cos(double a)`
- `tan(double a)`
- `acos(double a)`
- `asin(double a)`
- `atan(double a)`

Radians

- **Examples:**

```
Math.sin(0) returns 0.0
Math.sin(Math.PI / 6)
  returns 0.5
Math.sin(Math.PI / 2)
  returns 1.0
Math.cos(0) returns 1.0
Math.cos(Math.PI / 6)
  returns 0.866
Math.cos(Math.PI / 2)
  returns 0
```

Exponent Methods

- **`exp(double a)`**
  Returns `e` raised to the power of `a`.
- **`log(double a)`**
  Returns the natural logarithm of `a`.
- **`log10(double a)`**
  Returns the 10-based logarithm of `a`.
- **`pow(double a, double b)`**
  Returns `a` raised to the power of `b`.
- **`sqrt(double a)`**
  Returns the square root of `a`.

- **Examples:**

```
Math.exp(1) returns 2.71
Math.log(2.71)
   returns 1.0
Math.pow(2, 3)
   returns 8.0
Math.pow(3, 2)
   returns 9.0
Math.pow(3.5, 2.5)
   returns 22.91765
Math.sqrt(4) returns 2.0
Math.sqrt(10.5)
   returns 3.24
```

Rounding Methods

- **`double ceil(double x)`**

  x rounded up to its nearest integer. This integer is returned as a double value.

- **`double floor(double x)`**

  x is rounded down to its nearest integer. This integer is returned as a double value.

- **`double rint(double x)`**

  x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double.

- **`int round(float x)`**

  Return (int)Math.floor(x+0.5).

- **`long round(double x)`**

  Return (long)Math.floor(x+0.5).

Rounding Methods Examples

Math.ceil(2.1) returns 3.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.0) returns -2.0
Math.ceil(-2.1) returns -2.0
Math.floor(2.1) returns 2.0
Math.floor(2.0) returns 2.0
Math.floor(-2.0) returns -2.0
Math.floor(-2.1) returns -3.0
Math.round(2.6f) returns 3
Math.round(2.0) returns 2 (long)
Math.round(-2.0f) returns -2
Math.round(-2.6) returns -3 (long)

`min`, `max`, and `abs`

- **`max(a, b)`** and **`min(a, b)`**

  Returns the maximum or minimum of two parameters.

- **`abs(a)`**

  Returns the absolute value of the parameter.

- **`random()`**

  Returns a random `double` value in the range [0.0, 1.0).

- **Examples:**

```
Math.max(2, 3)
  returns 3
Math.max(2.5, 3)
  returns 3.0
Math.min(2.5,3.6)
  returns 2.5
Math.abs(-2)
  returns 2
Math.abs(-2.1)
  returns 2.1
```

The **random** Method

Generates a random **double** value greater than or equal to 0.0 and less than 1.0 (**0 <= Math.random() < 1.0**)

Examples:

```
(int)(Math.random() * 10)
```
→ Returns a random integer between 0 and 9.

```
50 + (int)(Math.random() * 50)
```
→ Returns a random integer between 50 and 99.

In general,

```
a + Math.random() * b
```
→ Returns a random number between a and a + b, excluding a + b.

Generating Random Characters

```
(char)((int)'a' + Math.random() * ((int)'z' - (int)'a' + 1))
```

- All numeric operators can be applied to the char operands
- The char operand is cast into a number if the other operand is a number or a character.
- So, the preceding expression can be simplified as follows:

```
(char)('a' + Math.random() * ('z' - 'a' + 1))
```

ASCII Code for Commonly Used Characters

Characters	Code Value in Decimal	Unicode Value
`'0'` to `'9'`	48 to 57	\u0030 to \u0039
`'A'` to `'Z'`	65 to 90	\u0041 to \u005A
`'a'` to `'z'`	97 to 122	\u0061 to \u007A

There is no need to remember them since we can do all mathematical operations with characters:

```
(char)('a' + Math.random() * ('z' - 'a' + 1))
```

```
'0' <= c && c <= '9'
```

Comparing and Testing Characters

```
if ('A' <= ch && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");

if ('a' <= ch && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");

if ('0' <= ch && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

Methods in the Character Class

Method	Description
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOrDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

The String Type

- The char type only represents one character:

```
char ch = 'a';
```

'a'

- To represent a string of characters, use the data type called String.

String is a predefined class in the Java library just like the System class

http://java.sun.com/javase/8/docs/api/java/lang/String.html

- The String type is NOT a primitive type.
  - The String type is a *reference type*.
    - A String variable is a reference variable, an "*address*" which points to an object storing the value or actual text

```
String message = "Welcome to Java";
```

More about Strings

- Each character is stored at an index:

```
String sentence = "A statement";
                  012345678910
```

- The String class API has methods to process strings:

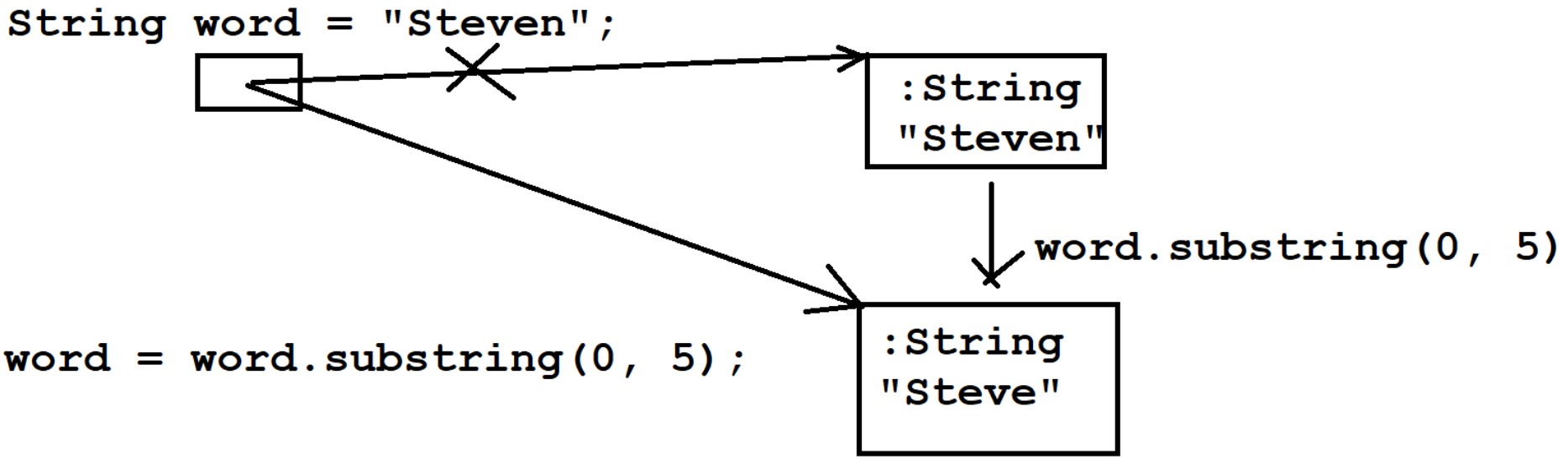
```
System.out.println("charAt(6) is " +
  sentence.charAt(6));
System.out.println(sentence.toUpperCase());
System.out.println(sentence.substring(0,7) +
  sentence.substring(10));
```

Strings are immutable!

- There are no methods to change them once they have been created
  - any new assignment will assign a new String reference to the old variable

```
String word = "Steven";
word = word.substring(0, 5);
```

  - the variable word is now a reference to a new String that contains "Steve"



String Concatenation

- "+" is used for making a new string by concatenating strings:

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";
// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2
// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B';
     // s1 becomes SupplementB
String s2 = 1 + 2 + "ABC";
     // s2 become "3ABC"
```

Special Characters

- `\n` – newline
- `\t` – tab
- `\"` – quotation mark
- Example:

```
System.out.print(s + "\n");
```

Reading a String from the Console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces:");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

Useful String functions

- **`charAt, equals, equalsIgnoreCase, compareTo, startsWith, endsWith, indexOf, lastIndexOf, replace, substring, toLowerCase, toUpperCase, trim`**

Comparing Strings

- Don't use '==' to compare Strings
  - it compares their memory addresses and not actual strings (character sequences)
- Instead use the **`equals`** method supplied by the String class:
  - **`s.equals(t)`**
    - **returns `true` if `s` and `t` have same letters and sequence**
    - **`false` otherwise**

Comparing Strings

```
String word1 = new String("Hello");
String word2 = new String("Hello");
if (word1 == word2){
  System.out.println(true);
} else {
  System.out.println(false);
}
```

false

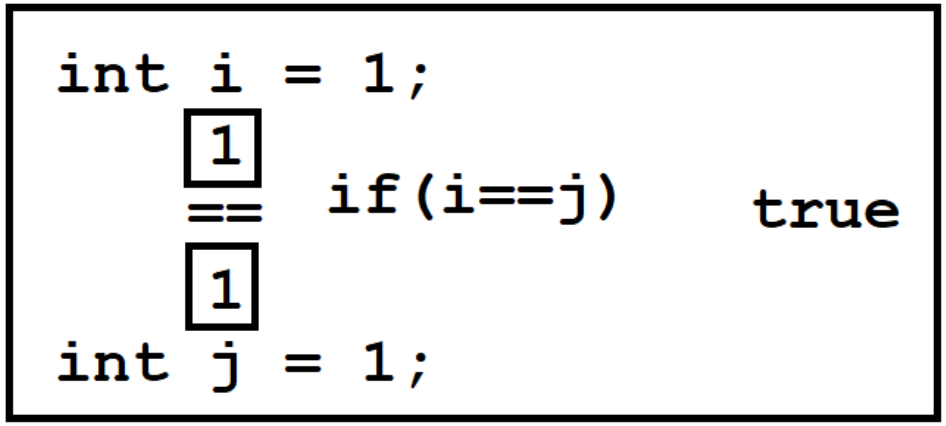
Two different addresses

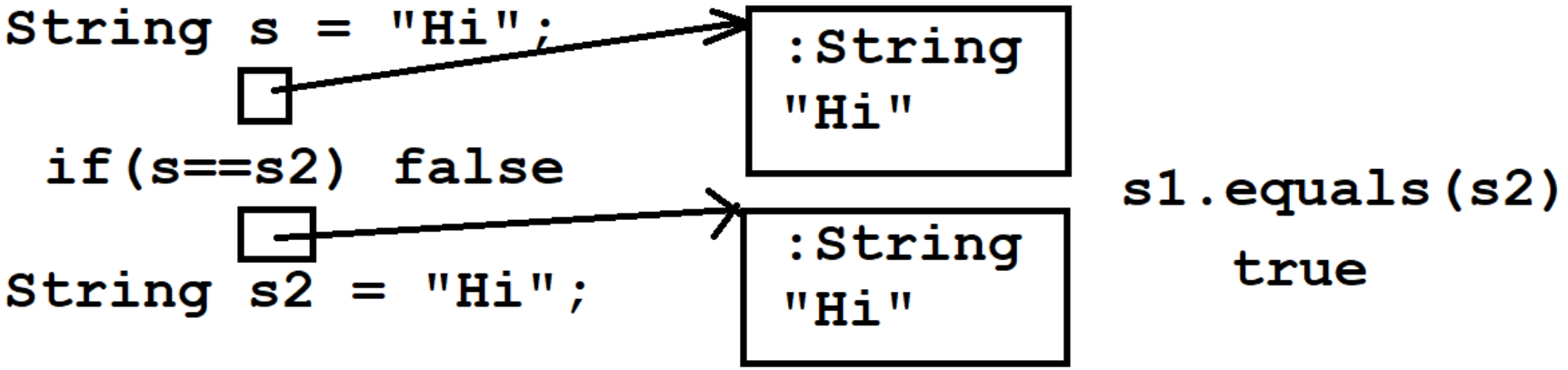
Comparing Strings

```
String word1 = new String("Hello");
String word2 = new String("Hello");
if (word1.equals(word2)){
  System.out.println(true);
} else {
  System.out.println(false);
}
```

true

compares the contents "Hello" with "Hello"





Comparing Strings

```
String word1 = "Hello";
String word2 = "Hello";
if (word1 == word2){
  System.out.println(true);
} else {
  System.out.println(false);
}
```

true

- Interned Strings: Only one instance of "Hello" is stored
  - so word1 and word2 will have the same address

Comparing Strings

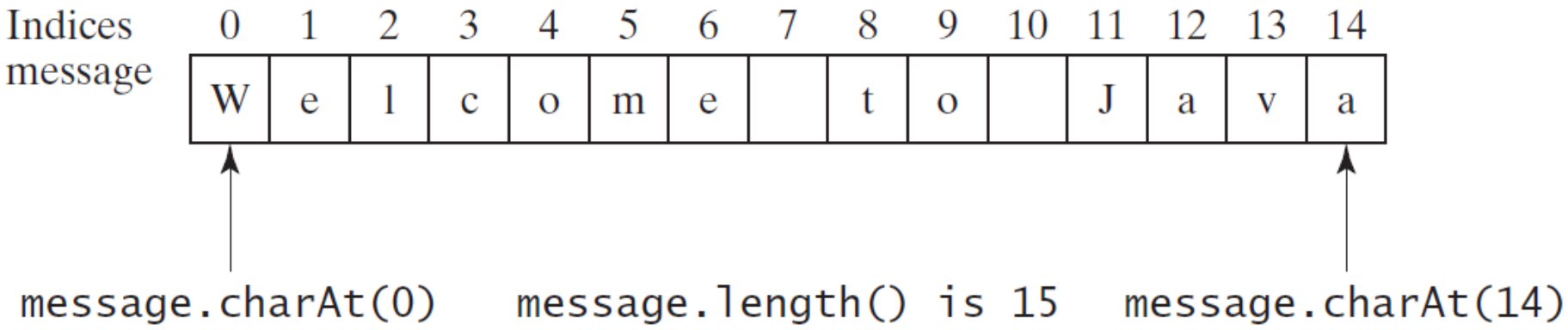
```
String word1 = "Hello";
String word2 = "Hello";
if (word1.equals(word2)){
  System.out.println(true);
} else {
  System.out.println(false);
}
```

true

Comparing Strings

Method	Description
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than `0`, equal to `0`, or less than `0` to indicate whether this string is greater than, equal to, or greater than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.

Getting Characters from a String




```
String message = "Welcome to Java";
System.out.println(
    "The first character in message is "
    + message.charAt(0));
```

Reading a single Character from the Console

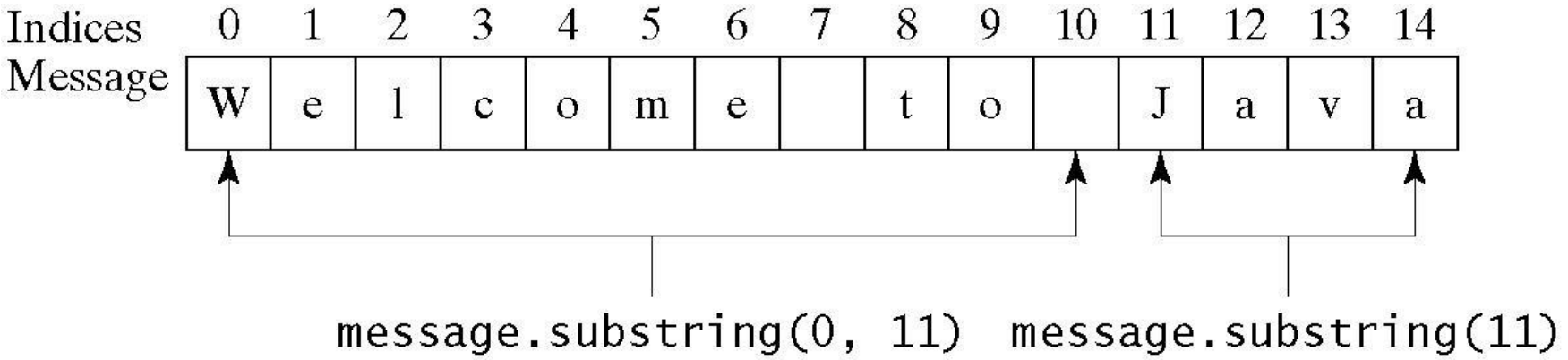
```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");

String s = input.nextLine();
char ch = s.charAt(0);

System.out.print("The character entered is "+ch);
```

Obtaining Substrings

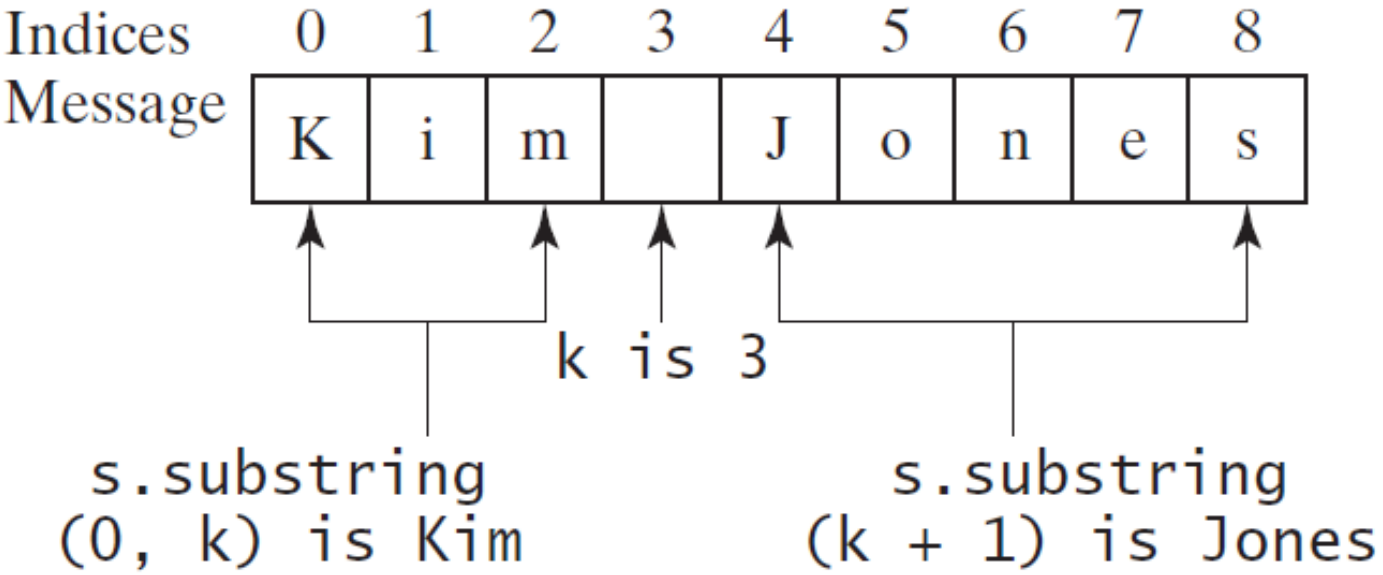
Method	Description
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, as shown in Figure 9.6. Note that the character at `endIndex` is not part of the substring.



Finding a Character or a Substring in a String

Method	Description
`indexOf(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

Finding a Character or a Substring in a String




```
int k = s.indexOf(' ');  //3
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```

Conversion between Strings and Numbers

```
String intString = "15";
String doubleString = "56.77653";

int intValue =
    Integer.parseInt(intString);
double doubleValue =
    Double.parseDouble(doubleString);

String s2 = "" + intValue;
```

Formatting Output

The printf statement:

```
System.out.printf(format, items);
```

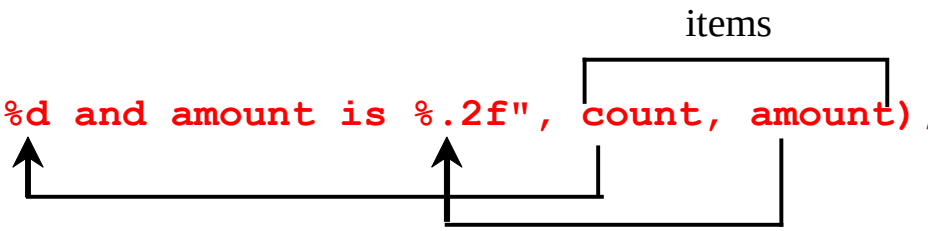
format is a string that may consist of substrings and format **specifiers**

- A format specifier begins with a percent sign and specifies how an item should be displayed: a numeric value, character, boolean value, or a string

Frequently-Used Specifiers

Specifier	Output	Example
%b	a boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s	a string	"Java is cool"

```
int count = 5;
double amount = 45.567899;
System.out.printf("count is %d and amount is %.2f", count, amount);
```



Displays: `count is 5 and amount is 45.56`

Bitwise operations in java

- To write programs at the machine-level, often you need to deal with binary numbers directly and perform operations at the bit-level
- Java provides the bitwise operators and shift operators
  - The bit operators apply only to integer types (byte, short, int, and long)
  - All bitwise operators can form bitwise assignment operators, such as **=**: **|=**, **<<=**, **>>=**, and **>>>=**
- Bitwise AND: &
  - 1010 & 1001 yields 1000

```
System.out.print(10&9); // 8
```

  - The AND of two corresponding bits yields a 1 if both bits are 1, otherwise 0

Bitwise operations in java

- Bitwise OR: |
  - The OR of two corresponding bits yields a 1 if either bit is 1
  - 10101110 | 10010010 yields 10111110

```
class BitwiseOR {
    public static void main(String[] args) {
     int number1 = 12, number2 = 25, result;
     result = number1 | number2;
     System.out.println(result);
    }
}
```

```
 1100 |   12
11001     25
11101     = 29
```

Bitwise operations in java

- Bitwise exclusive OR: ^
  - 1010 ^ 1001 yields 0011
  - The XOR of two corresponding bits yields a 1 only if two bits are different.
- One's complement: ~
  - ~1010 yields 0101
  - The operator toggles each bit from 0 to 1 and from 1 to 0.
- Left shift: <<
  - 1010 << 2 yields101000

```
System.out.print(10 << 2); // 40
```

  - The operator shifts bits in the first operand left by the number of bits specified in the second operand, filling with 0s on the right.

Bitwise operations in java

- Right shift with sign extension: >>
  - 1010 >> 2 yields 10

```
System.out.print(10 >> 2); // 2
```

  - The operator shifts bit in the first operand right by the number of bits specified in the second operand, filling with the highest (sign) bit on the left.
- Unsigned right shift with zero extension: >>>

```
System.out.print(-10 >>> 2); // 1073741821
```

  - The operator shifts bit in the first operand right by the number of bits specified in the second operand, filling with 0s on the left.

Constants in binary format

```
byte fourTimesThree = 0b1100;
byte data = 0b0000110011;
short number = 0b1111111111111111;
int overflow = 0b10101010101010101010101010101011;
long bow = 0b10101010101010101010101010101011L;
```

- Just be careful not to overflow the numbers with too much data, or else you'll get a compiler error:

```
byte data = 0b1100110011;
// Type mismatch: cannot convert from int to byte
```

- New feature in Java 7 known as numeric literals with underscores:

```
int overflow = 0b1010_1010_1010_1010_1010_1010_1010_1011;
long bow = 0b1__01010101__01010101__01010101__01010111L;
```

Constants in octal and hexadecimal format

```
int x = 010;       //octal = 8
int y = 0xf;      //hexadecimal = 15
```