# Chapter 4 Mathematical Functions, Characters, and Strings

- To solve mathematical problems by using the methods in the **Math** class
- To represent characters using the **char** type
  - To cast a numeric value to a character and cast a character to an integer
  - To compare and test characters using the static methods in the **Character** class
- To introduce objects and instance methods
- To represent strings using the **String** object
  - To return the string length using the **length()** method
  - To return a character in the string using the **charAt(i)** method
  - To read strings from the console
  - To read a character from the console
  - To compare strings using the **equals** and the **compareTo** methods
  - To find a character or a substring in a string using the **indexOf** method
- Case Studies
  - To program using characters and strings (GuessBirthday)
- To format output using the **System.out.printf** method

# The `Math` Class

Class constants:

- Math.PI
- Math.E : (the base of natural logarithm)

Class methods:

- Trigonometric Methods
- Exponent Methods
- Rounding Methods
- min, max, abs, and random Methods

# Trigonometric Methods

**sin**(double a)

**cos**(double a)

**tan**(double a)

**asin**(double a)

**acos**(double a)

**atan**(double a)

- The parameter for **sin**, **cos**, and **tan :**
- The return value for **asin**, **acos**, and **atan :**
  - is an angle in **radians**.

Examples:

Math.sin(0) returns 0.0

Math.sin(Math.PI / 6) returns 0.5

Math.asin(0.5) returns π/6

**Math.toDegrees**(Math.PI/2) returns 90.0

**Math.toRadians**(30) returns π/6

# 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.

```
Math.ceil(2.1) returns 3.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.1) returns -2.0
```

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

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

```
Math.floor(2.1) returns 2.0
Math.floor(-2.1) returns -3.0
```

`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.

```
Math.rint(2.1) returns 2.0
Math.rint(2.5) returns 2.0
```

# Rounding Methods

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

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

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

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

```
Math.round(2.4) returns 2
Math.round(2.6) returns 3 //returns long
Math.round(2.6f) returns 3 //returns int
```

# 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.abs(-2) returns 2

Math.abs(-2.1) returns 2.1

# The random Method

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.
```

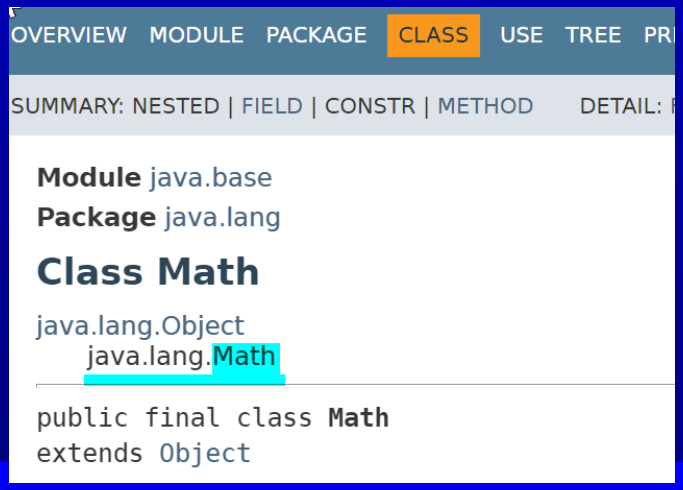
## Tip

You can view the complete documentation for the `Math` class online at http://java.sun.com/javase/6/docs/api/index.html, as shown in Figure 5.7.

## Note

Not all classes need a `main` method. The `Math` class and `JOptionPane` class do not have `main` methods. These classes contain methods for other classes to use.

https://docs.oracle.com/en/java/javase/



# Character Data Type

Four hexadecimal (base 16) digits.

char letter = 'A'; (ASCII)

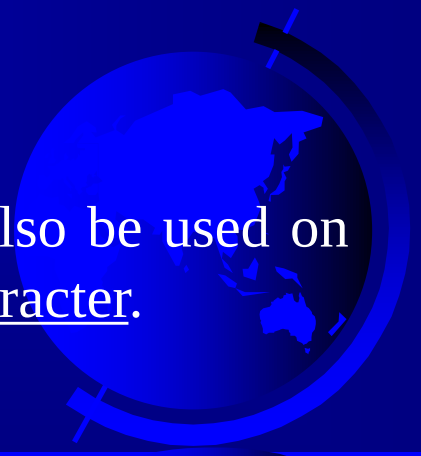
char numChar = '4'; (ASCII)

char letter = ‘\u0041'; (Unicode)

char numChar = '\u0034'; (Unicode)

```
char ch = 'a';
System.out.println(++ch);
```

– NOTE: The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character.



# ASCII Character Set, cont.

ASCII Character Set

is a subset of the Unicode from \u0000 to \u007f

TABLE B.2 ASCII Character Set in the Hexadecimal Index

| | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *A* | *B* | *C* | *D* | *E* | *F* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | nul | soh | stx | etx | eot | enq | ack | bel | bs | ht | nl | vt | ff | cr | so | si |
| **1** | dle | dcl | dc2 | dc3 | dc4 | nak | syn | etb | can | em | sub | esc | fs | gs | rs | us |
| **2** | sp | ! | “ | # | $ | % | & | ’ | ( | ) | * | + | , | - | . | / |
| **3** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| **4** | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| **5** | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| **6** | ‘ | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| **7** | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | del |

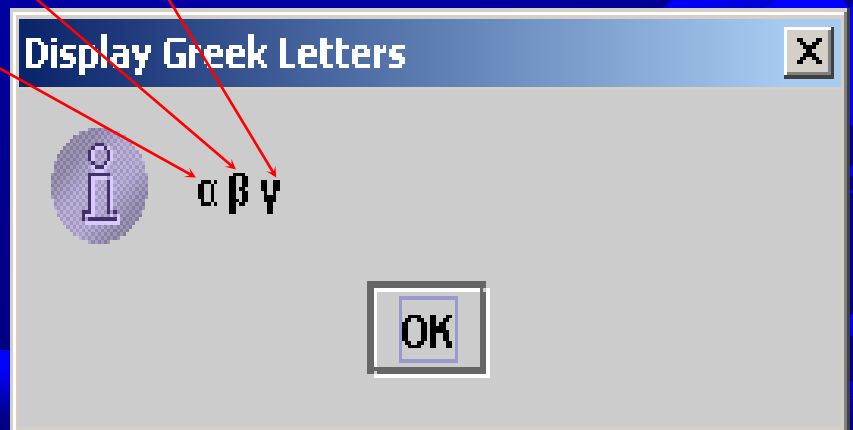
# Unicode Format

a 16-bit encoding scheme

Unicode takes two bytes, preceded by \u, expressed in four hexadecimal numbers

from '\u0000' to '\uFFFF‘ : `65535 + 1 characters`.

Unicode \u03b1 \u03b2 \u03b3 for three Greek letters



# Problem: Displaying with Unicode

Write a program that displays two Chinese characters and three Greek letters.

DisplayUnicode.java

```
1  import javax.swing.JOptionPane;
2
3  public class DisplayUnicode {
4    public static void main(String[] args) {
5      JOptionPane.showMessageDialog(null,
6        "\u6B22\u8FCE \u03b1 \u03b2 \u03b3",
7        "\u6B22\u8FCE Welcome",
8        JOptionPane.INFORMATION_MESSAGE);
9    }
10 }
```


If no Chinese font is installed on your system, you will not be able to see the Chinese characters. The Unicodes for the Greek letters $\alpha$ $\beta$ $\gamma$ are \u03b1 \u03b2 \u03b3.

# Casting between char and Numeric Types

```
int i = 'a'; // Same as int i = (int)'a';

char c = 97; // Same as char c = (char)97;
```

# The `Character` Class

- java.lang package provides a wrapper class for every primitive data type
  - enable the primitive data values to be treated as objects.
  - contain useful methods for processing primitive values.

| java.lang.Character | |
|---|---|
| +Character(value: char) | Constructs a character object with char value |
| +charValue(): char | Returns the char value from this object |
| +compareTo(anotherCharacter: Character): int | Compares this character with another |
| +equals(anotherCharacter: Character): boolean | Returns true if this character equals to another |
| +isDigit(ch: char): boolean | Returns true if the specified character is a digit |
| +isLetter(ch: char): boolean | Returns true if the specified character is a letter |
| +isLetterOrDigit(ch: char): boolean | Returns true if the character is a letter or a digit |
| +isLowerCase(ch: char): boolean | Returns true if the character is a lowercase letter |
| +isUpperCase(ch: char): boolean | Returns true if the character is an uppercase letter |
| +toLowerCase(ch: char): char | Returns the lowercase of the specified character |
| +toUpperCase(ch: char): char | Returns the uppercase of the specified character |

# Examples

**Character charObject = *new Character('b');***

**charObject.equals**(new Character('b')) returns true
**charObject.**equals(new Character('d')) returns false
**charObject.compareTo**(new Character('a')) returns 1
**charObject.**compareTo(new Character('b')) returns 0
**charObject.**compareTo(new Character('c')) returns -1
**charObject.**compareTo(new Character('d') returns –2

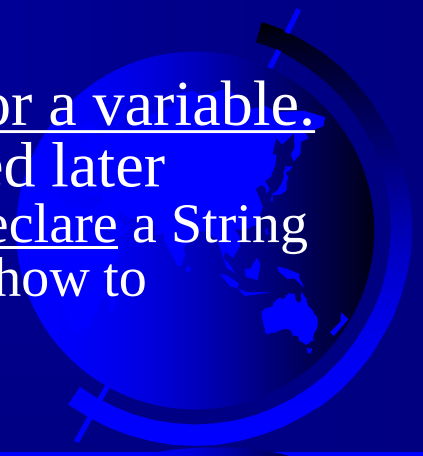
# The String Type/Class

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

String is actually a predefined class in the Java library just like the System class and JOptionPane class.

The String type is not a primitive type,
but a *reference type*.

- Any Java class can be used as a reference type for a variable.
- Reference data types will be thoroughly discussed later
  - For the time being, you just need to know how to declare a String variable, how to assign a string to the variable, and how to concatenate strings.



# String Concatenation

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

String message += " world" ;


// concatenated with a number
String s = “Chapter” + 2; // s becomes “Chapter2”

// concatenated with a character
String s1 = “Supplement” + ‘B’; // s1 becomes “SupplementB”
```

# String Comparisons

**equals**

```
String s1 = "Welcome";
String s2 = new String("Welcome");

if (s1.equals(s2)){
  // s1 and s2 have the same contents
}

if (s1 == s2) {
  // s1 and s2 have the same reference
 }
```

# compareTo(Object object)

```
String s1 = new String("Welcome");
String s2 = "Welcome";

if (s1.compareTo(s2) > 0) {
  // s1 is greater than s2
}
else if (s1.compareTo(s2) == 0) {
  // s1 and s2 have the same contents
}
else
   // s1 is less than s2
```

Return value is the Unicode offset of the first two distinct characters in **s1** and **s2** from left to right.

- For example, suppose **s1** is **"abc"** and **s2** is **"abg"**, **s1.compareTo(s2)** returns **-4**.

# String Comparisons

| java.lang.String | |
|---|---|
| +equals(s1: String): boolean | Returns true if this string is equal to string s1. |
| +equalsIgnoreCase(s1: String): boolean | Returns true if this string is equal to string s1 case insensitive. |
| +compareTo(s1: String): int | Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1. |
| +compareToIgnoreCase(s1: String): int | Same as compareTo except that the comparison is case insensitive. |
| +regionMatches(index: int, s1: String, s1Index: int, len: int): boolean | Returns true if the specified subregion of this string exactly matches the specified subregion in string s1. |
| +regionMatches(ignoreCase: boolean, index: int, s1: String, s1Index: int, len: int): boolean | Same as the preceding method except that you can specify whether the match is case sensitive. |
| +startsWith(prefix: String): boolean | Returns true if this string starts with the specified prefix. |
| +endsWith(suffix: String): boolean | Returns true if this string ends with the specified suffix. |

The `String` class contains the methods for comparing strings.

# String Length, Characters, and Combining Strings

| java.lang.String |
| --- |
| +length(): int |
| +charAt(index: int): char |
| +concat(s1: String): String |

Returns the number of characters in this string.

Returns the character at the specified index from this string.

Returns a new string that concatenate this string with string s1.

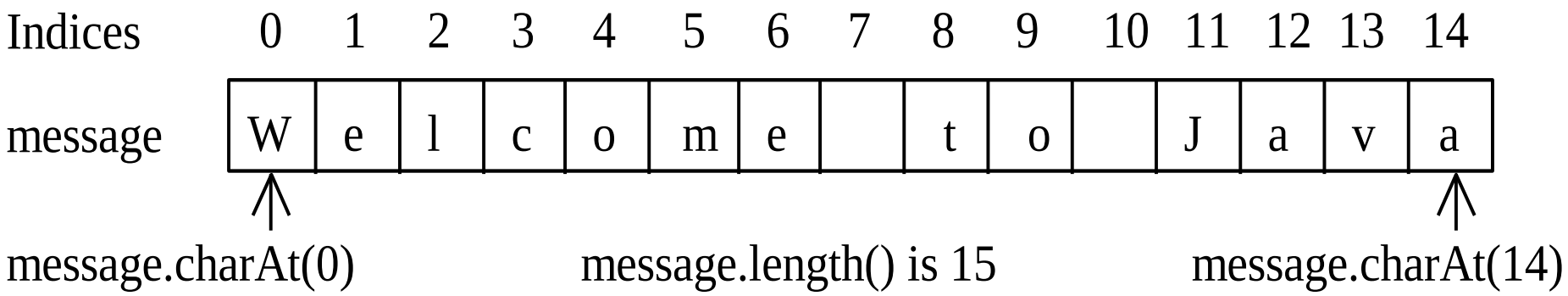
# Finding String Length

```
message = "Welcome";
```

```
message.length()
```
(returns 7)

# Retrieving Individual Characters in a String

- **Do not use `message[0]`**
- **Use `message.charAt(index)`**
  - **Index starts from `0`**



# String Concatenation

```
String s3 = s1.concat(s2);
```

**String s3 = s1 + s2;**

**(((s1.concat(s2)).concat(s3)).concat(s4)).concat(s5);**

same as

**s1 + s2 + s3 + s4 + s5**

# Extracting Substrings

**java.lang.String**

`+substring(beginIndex: int): String`

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 9.6.

`+substring(beginIndex: int, endIndex: int): String`

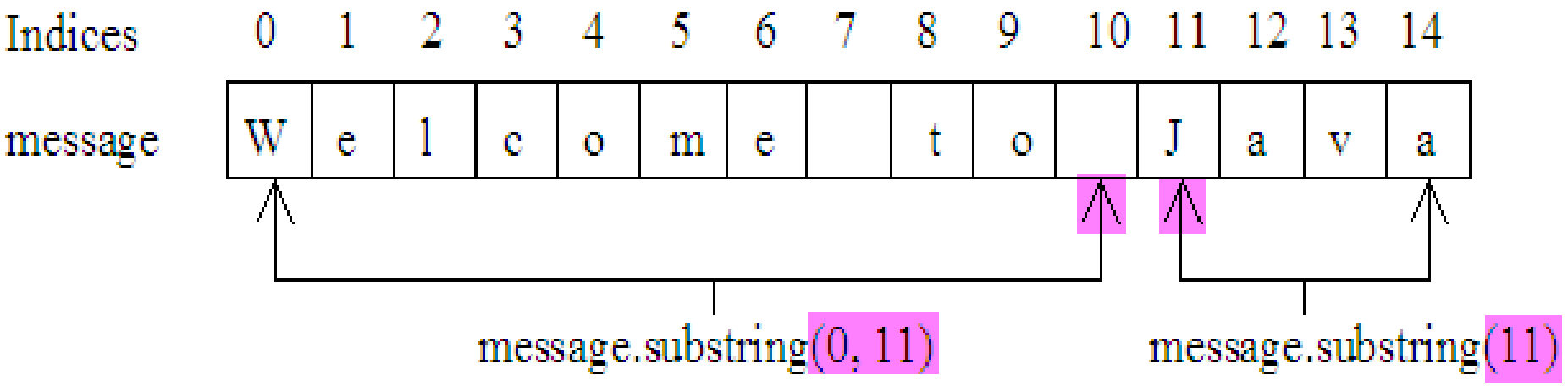
Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, as shown in Figure 9.6. Note that the character at `endIndex` is not part of the substring.

# Extracting Substrings

**extract** a single **character** from a string : **charAt** ()

**extract** a **substring** from a string : **substring** () method in the **String** class.

```
String s1 = "Welcome to Java";
String s2 = s1.substring(0, 11) + "HTML";
```



# Finding a Character or a Substring in a String

The String class provides several overloaded methods:

| java.lang.String | |
|---|---|
| +indexOf(ch: char): int | Returns the index of the first occurrence of ch in the string. Returns -1 if not matched. |
| +indexOf(ch: char, fromIndex: int): int | Returns the index of the first occurrence of ch after fromIndex in the string. Returns -1 if not matched. |
| +indexOf(s: String): int | Returns the index of the first occurrence of string s in this string. Returns -1 if not matched. |
| +indexOf(s: String, fromIndex: int): int | Returns the index of the first occurrence of string s in this string after fromIndex. Returns -1 if not matched. |
| +lastIndexOf(ch: int): int | Returns the index of the last occurrence of ch in the string. Returns -1 if not matched. |
| +lastIndexOf(ch: int, fromIndex: int): int | Returns the index of the last occurrence of ch before fromIndex in this string. Returns -1 if not matched. |
| +lastIndexOf(s: String): int | Returns the index of the last occurrence of string s. Returns -1 if not matched. |
| +lastIndexOf(s: String, fromIndex: int): int | 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

indexOf

```
"Welcome to Java".indexOf('W') returns 0.
"Welcome to Java".indexOf('o') returns 4.
"Welcome to Java".indexOf('o', 5) returns 9.
"Welcome to Java".indexOf("come") returns 3.
"Welcome to Java".indexOf("Java", 5) returns 11.
"Welcome to Java".indexOf("java", 5) returns -1.
```

lastIndexOf

```
"Welcome to Java".lastIndexOf('W') returns 0.
"Welcome to Java".lastIndexOf('o') returns 9.
"Welcome to Java".lastIndexOf('o', 5) returns 4.
"Welcome to Java".lastIndexOf("come") returns 3.
"Welcome to Java".lastIndexOf("Java", 5) returns -1.
"Welcome to Java".lastIndexOf("Java") returns 11.
```

# Convert Character and Numbers to Strings

String class provides overloaded static *valueOf* methods for converting a character, an array of characters, and numeric values to strings.

| java.lang.String | |
|---|---|
| +valueOf(c: char): String | Returns a string consisting of the character c. |
| +valueOf(data: char[]): String | Returns a string consisting of the characters in the array. |
| +valueOf(d: double): String | Returns a string representing the double value. |
| +valueOf(f: float): String | Returns a string representing the float value. |
| +valueOf(i: int): String | Returns a string representing the int value. |
| +valueOf(l: long): String | Returns a string representing the long value. |
| +valueOf(b: boolean): String | Returns a string representing the boolean value. |

For example, String.valueOf(5.44). the return value is string "5.44"

# Convert Strings to Numbers

convert a string to a **double** value

***Double.parseDouble*(str)**

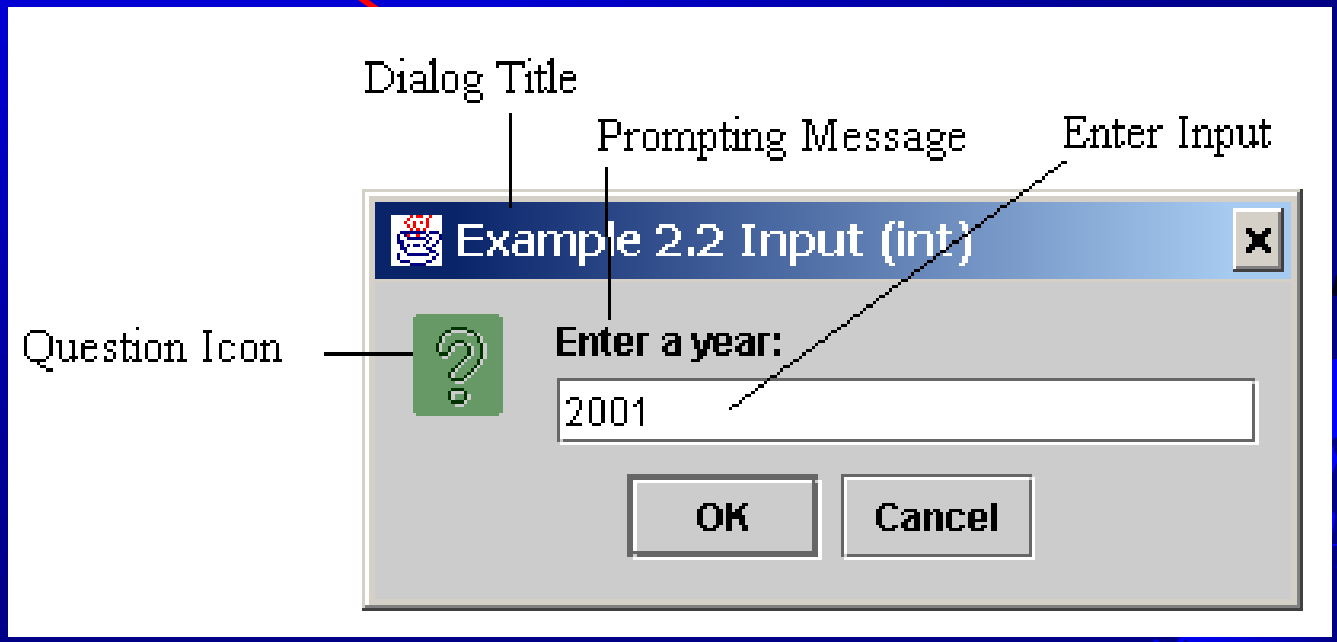
convert a string to an **int** value.

***Integer.parseInt*(str)**

The `Integer` and `Double` classes are both included in the `java.lang` package, and thus they are automatically imported.

# Review: Getting Input from Input Dialog Boxes

```
String input = JOptionPane.showInputDialog(
  "Enter a year:");
```



# Review: Converting Strings to Integers

The the input dialog box returns a String "2001".

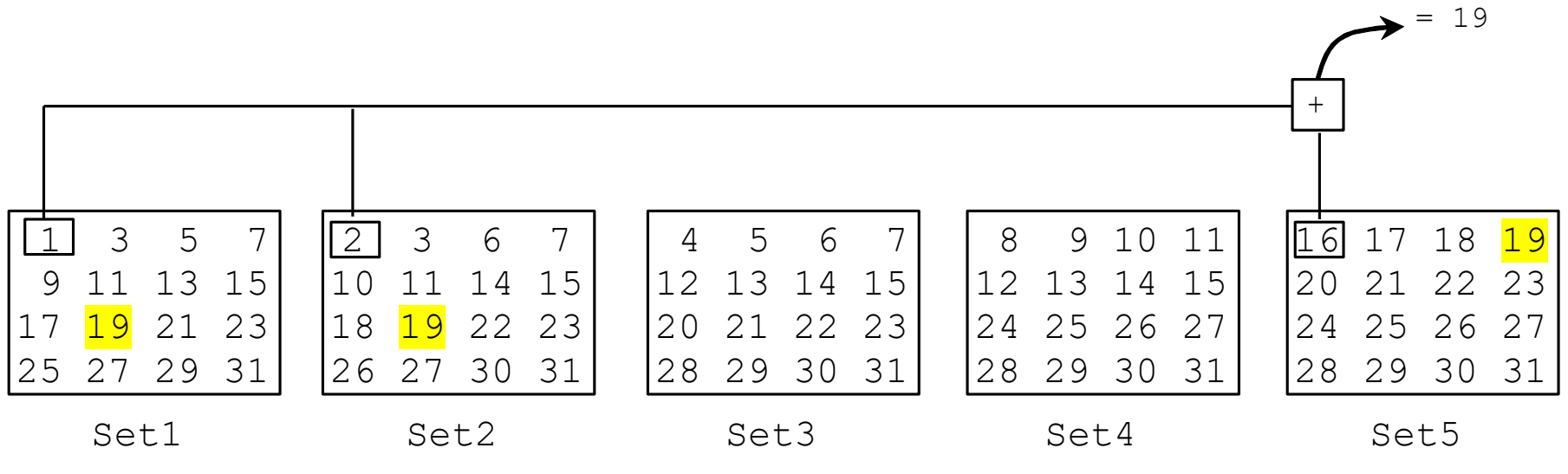
To obtain the input as a number, you have to convert a string into a number.

```
int intValue = Integer.parseInt(intString);
```

# Problem: Guessing Birthday

The program can guess your birth date (the date of the month) by asking five questions.

– Each question asks whether the day is in one of the five sets of numbers.




The birthday is the sum of the first numbers in the sets where the day appears. For example, if the birthday is 19, it appears in Set1, Set2, and Set5. The first numbers in these three sets are 1, 2, and 16. Their sum is 19.

```
Is your birthday in Set1?
 1  3  5  7
 9 11 13 15
17 19 21 23
25 27 29 31
Enter 0 for No and 1 for Yes: 1 Enter
Is your birthday in Set2?
 2  3  6  7
10 11 14 15
18 19 22 23
26 27 30 31
Enter 0 for No and 1 for Yes: 1 Enter
Is your birthday in Set3?
 4  5  6  7
12 13 14 15
20 21 22 23
28 29 30 31
Enter 0 for No and 1 for Yes: 0 Enter
Is your birthday in Set4?
 8  9 10 11
12 13 14 15
24 25 26 27
28 29 30 31
Enter 0 for No and 1 for Yes: 0 Enter
Is your birthday in Set5?
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
Enter 0 for No and 1 for Yes: 1 Enter
Your birthday is 19
```

# Mathematics Basis for the Game

The game is easy to program. You may wonder how the game was created. The mathematics behind the game is actually quite simple. The numbers are not grouped together by accident. The way they are placed in the five sets is deliberate. The starting numbers in the five sets are 1, 2, 4, 8, and 16, which correspond to 1, 10, 100, 1000, and 10000 in binary. A binary number for decimal integers between 1 and 31 has at most five digits, as shown in Figure 3.2(a). Let it be $b_5b_4b_3b_2b_1$. So, $b_5b_4b_3b_2b_1 = b_5\,0000 + b_4\,000 + b_3\,00 + b_2\,0 + b_1$, as shown in Figure 3.2(b). If a day's binary number has a digit 1 in $b_k$, the number should appear in Set$k$. For example, number 19 is binary 10011, so it appears in Set1, Set2, and Set5. It is binary 1 + 10 + 10000 = 10011 or decimal 1 + 2 + 16 = 19. Number 31 is binary 11111, so it appears in Set1, Set2, Set3, Set4, and Set5. It is binary 1 + 10 + 100 + 1000 + 10000 = 11111 or decimal 1 + 2 + 4 + 8 + 16 = 31.

| Decimal | Binary |
|---|---|
| 1 | 00001 |
| 2 | 00010 |
| 3 | 00011 |
| ... | |
| 19 | 10011 |
| ... | |
| 31 | 11111 |

(a)

$$\begin{array}{r} b_5\,0\,0\,0\,0 \\ b_4\,0\,0\,0 \\ b_3\,0\,0 \\ b_2\,0 \\ +\quad b_1 \\ \hline b_5\,b_4\,b_3\,b_2\,b_1 \end{array} \qquad \begin{array}{r} 10000 \\ 10 \\ +\quad 1 \\ \hline 10011 \end{array} \qquad \begin{array}{r} 10000 \\ 1000 \\ 100 \\ 10 \\ +\quad 1 \\ \hline 11111 \end{array}$$

19 31

(b)

# Formatting Output

If you wish to display only two digits after the decimal point in a floating-point value, you may write the code like this:

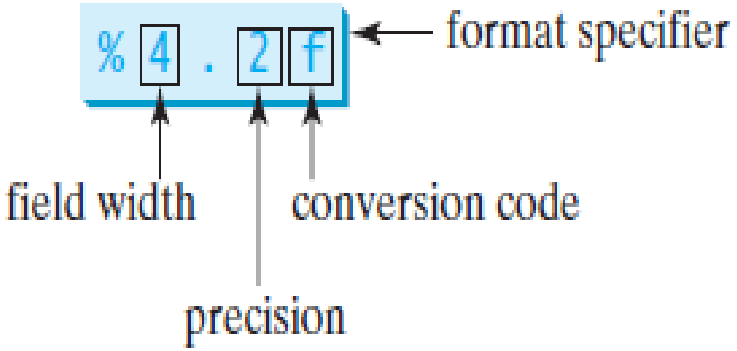
```
double x = 2.0 / 3;
System.out.println("x is " + (int)(x * 100) / 100.0);
```

```
x is 0.66
```

However, a better way to accomplish this task is to format the output using the `printf`

```
double x = 2.0 / 3;
System.out.printf("x is %4.2f", x);
```

display x is 0.67



# Formatting Output

Use the printf statement.

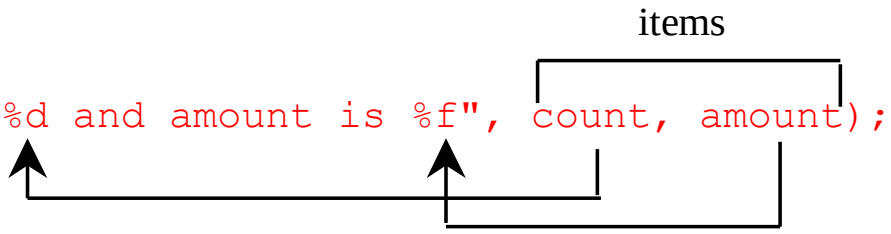
System.out.printf(format, items);

- format ： How to display.
  - each specifier begins with a percent sign %.
- item: What to display
  - 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 = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```




```
display              count is 5 and amount is 45.560000
```

**Tip**

The % sign denotes a specifier. To output a literal % in the format string, use %%.

## Examples of Specifying Width and Precision

| *Example* | *Output* |
|---|---|
| `%5c` | Output the character and add four spaces before the character item. |
| `%6b` | Output the Boolean value and add one space before the false value and two spaces before the true value. |
| `%5d` | Output the integer item with width at least 5. If the number of digits in the item is $<$ 5, add spaces before the number. If the number of digits in the item is $>$ 5, the width is automatically increased. |
| `%10.2f` | Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus there are 7 digits allocated before the decimal point. If the number of digits before the decimal point in the item is $<$ 7, add spaces before the number. If the number of digits before the decimal point in the item is $>$ 7, the width is automatically increased. |
| `%10.2e` | Output the floating-point item with width at least 10 including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than 10, add spaces before the number. |
| `%12s` | Output the string with width at least 12 characters. If the string item has less than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased. |

```
double x = 2.0 / 3;
System.out.printf("x is %4.2f", x);
display              x is 0.67
```