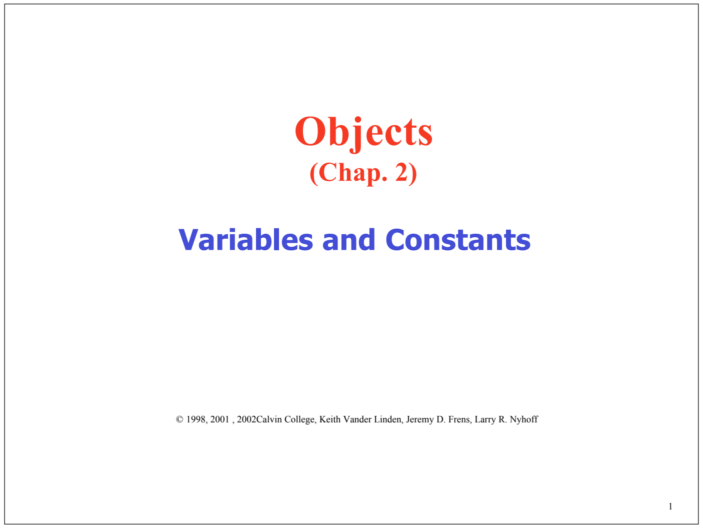
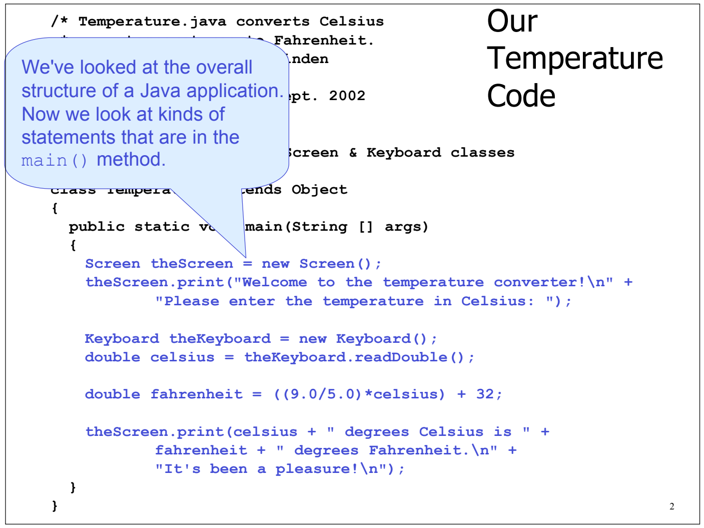


Objects

(Chap. 2)

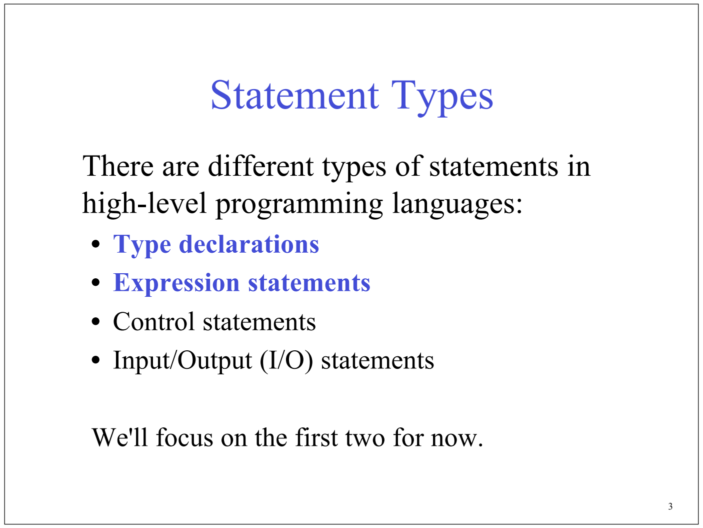
Variables and Constants



Our Temperature Code

```
/* Temperature.java converts Celsius
                         Fahrenheit.
                         inden

                         pt. 2002


                         creen & Keyboard classes

class Tempera           tends Object
{
  public static vo     main(String [] args)
  {
    Screen theScreen = new Screen();
    theScreen.print("Welcome to the temperature converter!\n" +
            "Please enter the temperature in Celsius: ");

    Keyboard theKeyboard = new Keyboard();
    double celsius = theKeyboard.readDouble();

    double fahrenheit = ((9.0/5.0)*celsius) + 32;

    theScreen.print(celsius + " degrees Celsius is " +
            fahrenheit + " degrees Fahrenheit.\n" +
            "It's been a pleasure!\n");
  }
}
```

We've looked at the overall structure of a Java application. Now we look at kinds of statements that are in the `main()` method.

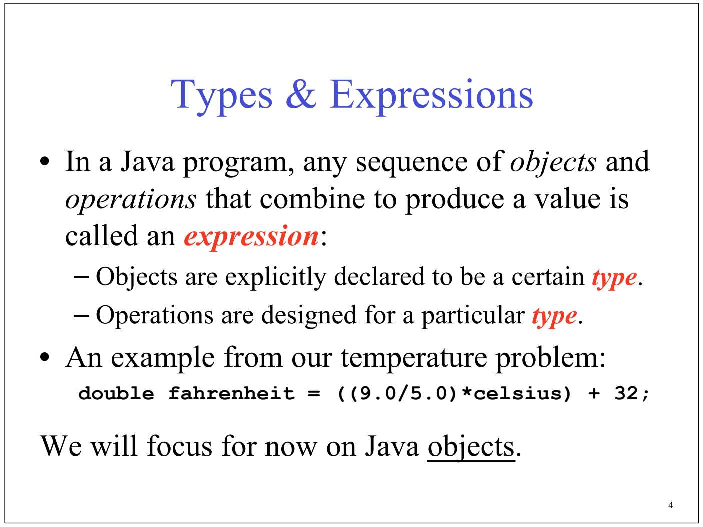


Statement Types

There are different types of statements in high-level programming languages:

- **Type declarations**
- **Expression statements**
- Control statements
- Input/Output (I/O) statements

We'll focus on the first two for now.

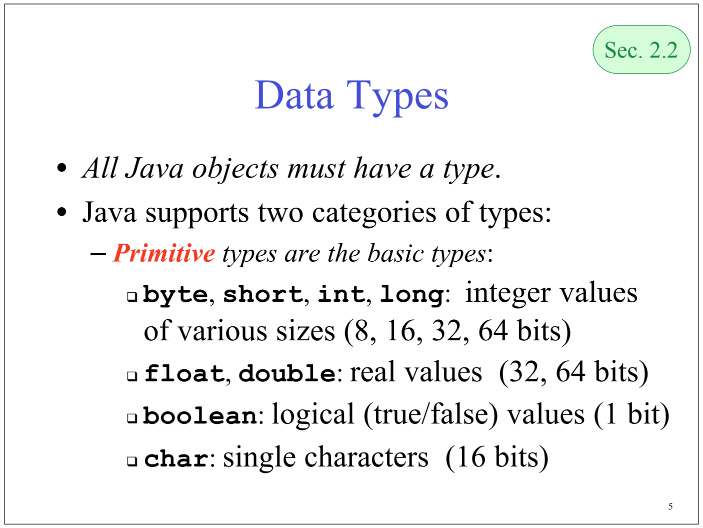


Types & Expressions

- In a Java program, any sequence of *objects* and *operations* that combine to produce a value is called an ***expression***:
  - Objects are explicitly declared to be a certain ***type***.
  - Operations are designed for a particular ***type***.
- An example from our temperature problem:

```
double fahrenheit = ((9.0/5.0)*celsius) + 32;
```

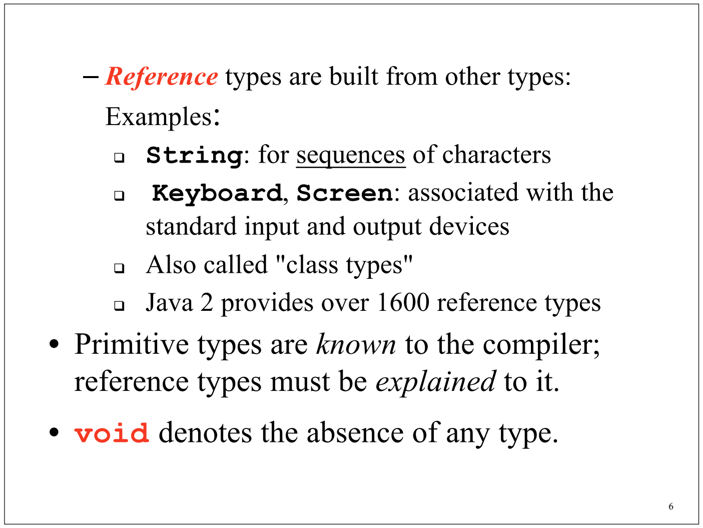
We will focus for now on Java <u>objects</u>.



Sec. 2.2

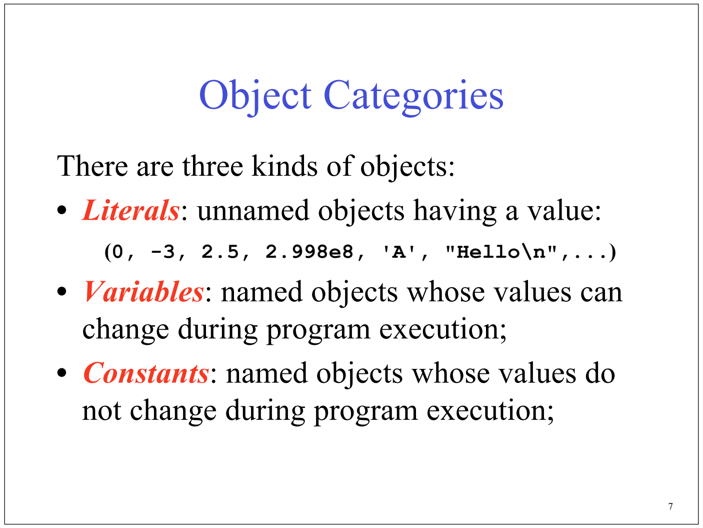
Data Types

- *All Java objects must have a type.*
- Java supports two categories of types:
  - *Primitive types are the basic types*:
    - ❑ **`byte`**, **`short`**, **`int`**, **`long`**: integer values of various sizes (8, 16, 32, 64 bits)
    - ❑ **`float`**, **`double`**: real values (32, 64 bits)
    - ❑ **`boolean`**: logical (true/false) values (1 bit)
    - ❑ **`char`**: single characters (16 bits)



- *Reference* types are built from other types:

  Examples:
  - ❑ **`String`**: for sequences of characters
  - ❑ **`Keyboard`**, **`Screen`**: associated with the standard input and output devices
  - ❑ Also called "class types"
  - ❑ Java 2 provides over 1600 reference types
- Primitive types are *known* to the compiler; reference types must be *explained* to it.
- **`void`** denotes the absence of any type.

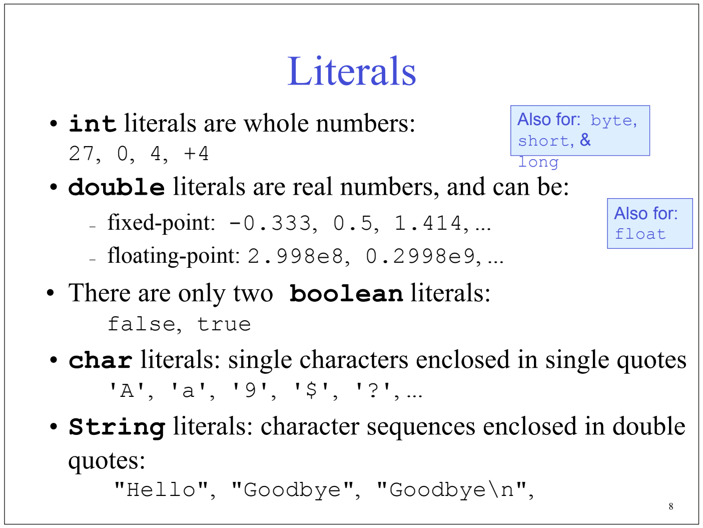


Object Categories

There are three kinds of objects:

- *Literals*: unnamed objects having a value:

  `(0, -3, 2.5, 2.998e8, 'A', "Hello\n",...)`
- *Variables*: named objects whose values can change during program execution;
- *Constants*: named objects whose values do not change during program execution;



Literals

Also for: `byte`, `short`, & `long`

Also for: `float`

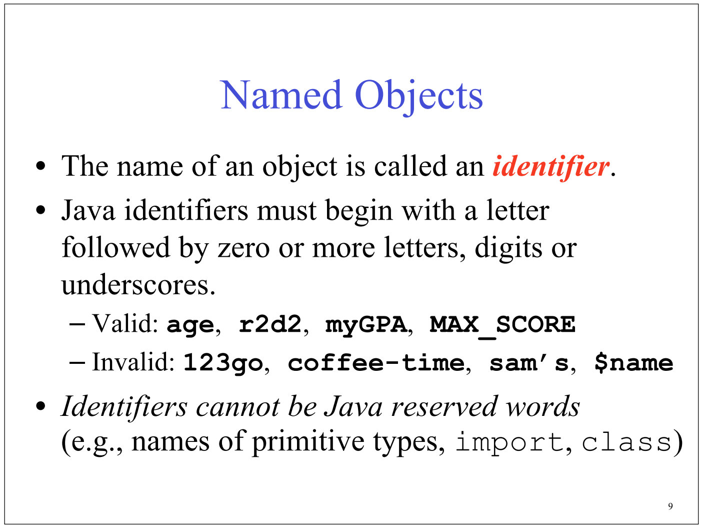
- **`int`** literals are whole numbers:

  `27, 0, 4, +4`
- **`double`** literals are real numbers, and can be:
  - fixed-point: `-0.333, 0.5, 1.414`, ...
  - floating-point: `2.998e8, 0.2998e9`, ...
- There are only two **`boolean`** literals:

  `false, true`
- **`char`** literals: single characters enclosed in single quotes

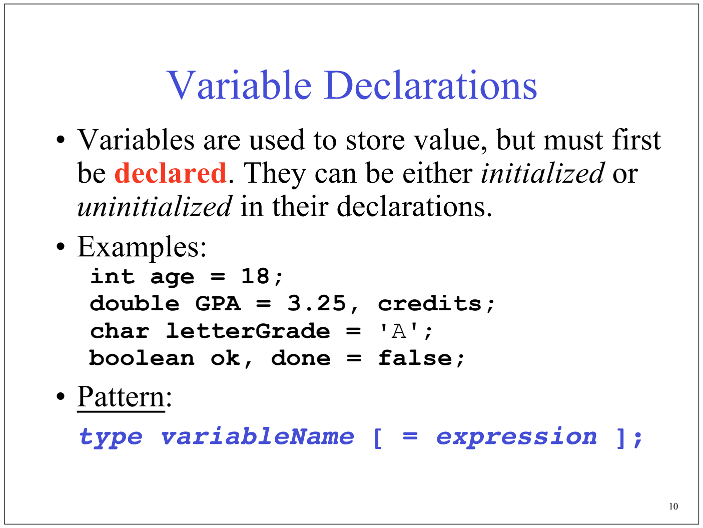
  `'A', 'a', '9', '$', '?'`, ...
- **`String`** literals: character sequences enclosed in double quotes:

  `"Hello", "Goodbye", "Goodbye\n",`



Named Objects

- The name of an object is called an ***identifier***.
- Java identifiers must begin with a letter followed by zero or more letters, digits or underscores.
  - Valid: **`age`**, **`r2d2`**, **`myGPA`**, **`MAX_SCORE`**
  - Invalid: **`123go`**, **`coffee-time`**, **`sam's`**, **`$name`**
- *Identifiers cannot be Java reserved words* (e.g., names of primitive types, `import`, `class`)



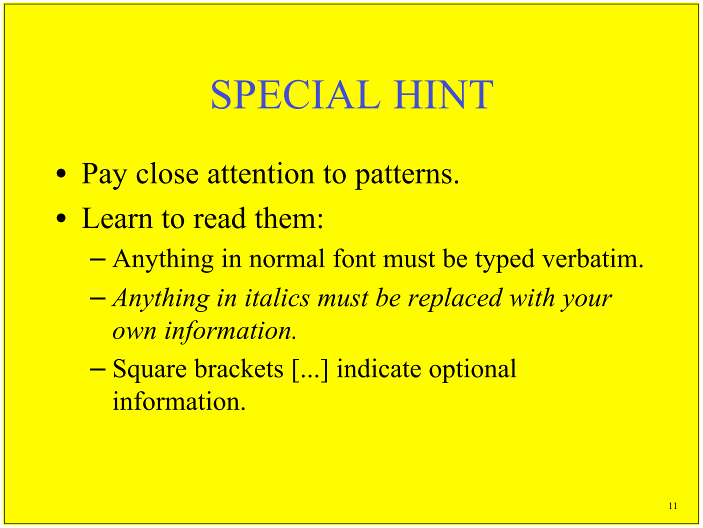
Variable Declarations

- Variables are used to store value, but must first be **declared**. They can be either *initialized* or *uninitialized* in their declarations.
- Examples:

```
int age = 18;
double GPA = 3.25, credits;
char letterGrade = 'A';
boolean ok, done = false;
```

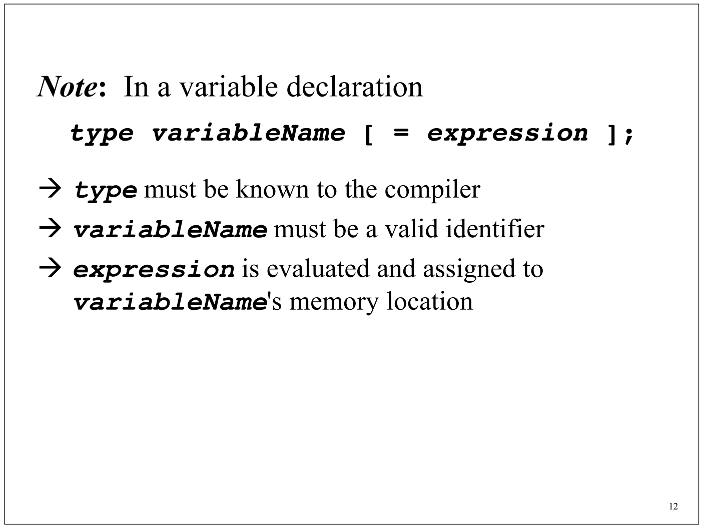
- Pattern:

type variableName **[= *expression*];**



SPECIAL HINT

- Pay close attention to patterns.
- Learn to read them:
  - Anything in normal font must be typed verbatim.
  - *Anything in italics must be replaced with your own information.*
  - Square brackets [...] indicate optional information.



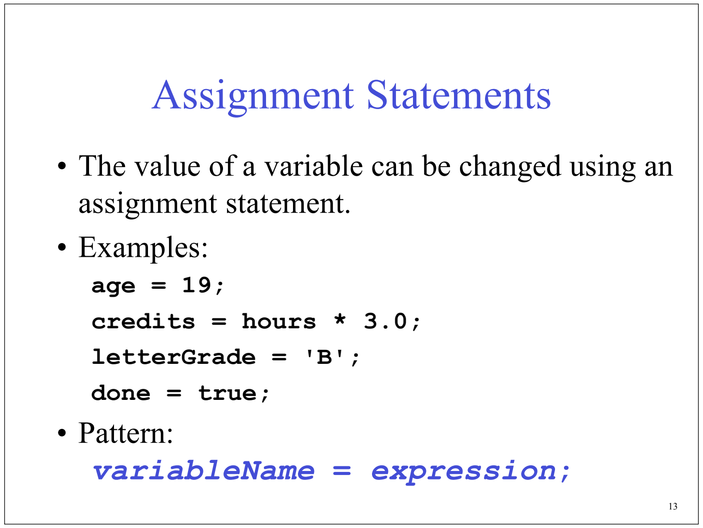
***Note*:** In a variable declaration

type variableName **[= *expression*];**

→ ***type*** must be known to the compiler

→ ***variableName*** must be a valid identifier

→ ***expression*** is evaluated and assigned to ***variableName***'s memory location



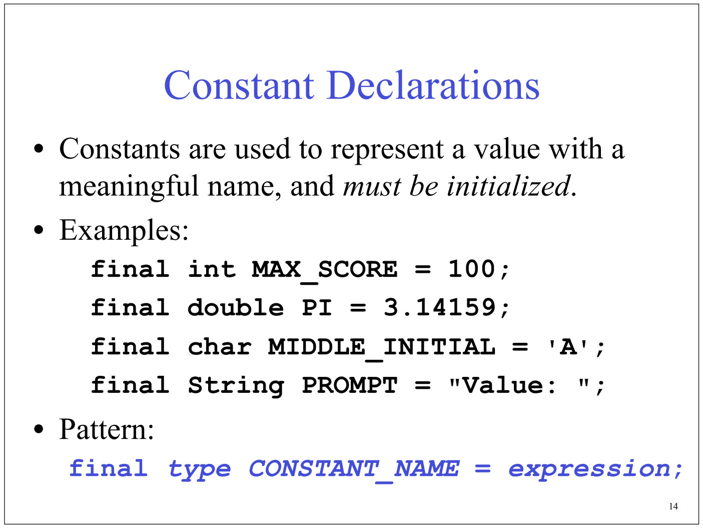
Assignment Statements

- The value of a variable can be changed using an assignment statement.
- Examples:

```
age = 19;
credits = hours * 3.0;
letterGrade = 'B';
done = true;
```

- Pattern:

```
variableName = expression;
```



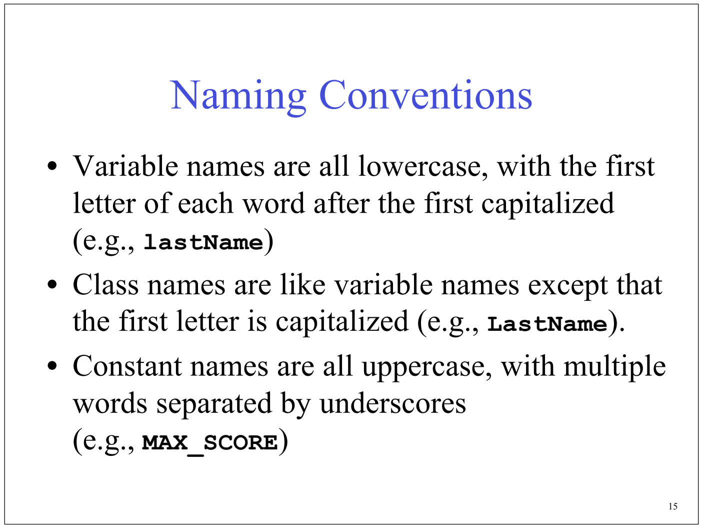
Constant Declarations

- Constants are used to represent a value with a meaningful name, and *must be initialized*.
- Examples:

```
final int MAX_SCORE = 100;
final double PI = 3.14159;
final char MIDDLE_INITIAL = 'A';
final String PROMPT = "Value: ";
```

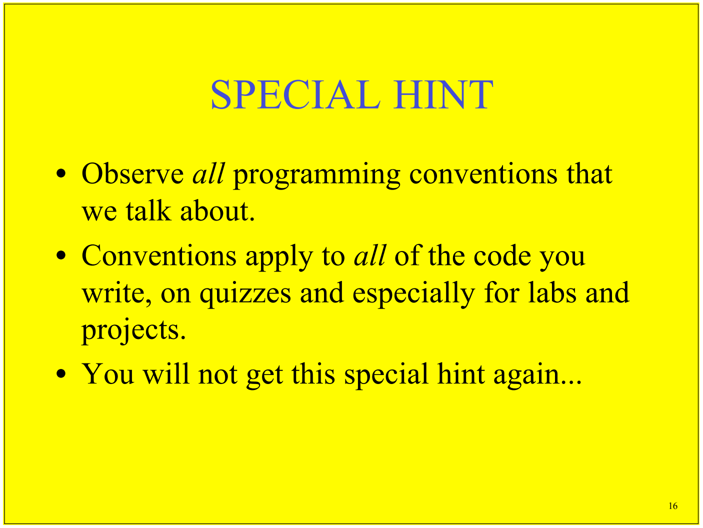
- Pattern:

```
final type CONSTANT_NAME = expression;
```



Naming Conventions

- Variable names are all lowercase, with the first letter of each word after the first capitalized (e.g., `lastName`)
- Class names are like variable names except that the first letter is capitalized (e.g., `LastName`).
- Constant names are all uppercase, with multiple words separated by underscores (e.g., `MAX_SCORE`)



SPECIAL HINT

- Observe *all* programming conventions that we talk about.
- Conventions apply to *all* of the code you write, on quizzes and especially for labs and projects.
- You will not get this special hint again...

Part of the Picture:
Data Representation

How literals of the primitive types are represented and stored in memory.

Representing Integers

Integers are often represented in the *twos-complement* format, where the high-order bit indicates the number's sign:

```
2₁₀  = 0000000000000010₂
1₁₀  = 0000000000000001₂
0₁₀  = 0000000000000000₂
-1₁₀ = 1111111111111111₂
-2₁₀ = 1111111111111110₂
```

What's going on here and why?

These examples have 16 bits, but 32 or 64 are more common.

Two's-Complement

For nonnegative n:
Use ordinary base-two representation with leading (sign) bit 0

For negative n (–n):
1. Find w-bit base-2 representation of n
2. Complement each bit.
3. Add 1

Example: –88

1. 88 as a 16-bit base-two number	0000000001011000
2. Complement this bit string	1111111110100111
3. Add 1	1111111110101000

1	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0

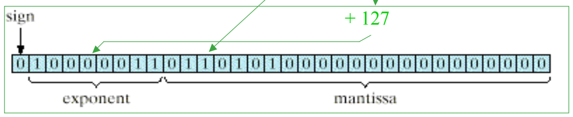
Shortcut for Step 3: Flip all bits from rightmost **0** to the end

Real Objects

Real values are often represented in 64 bits using the IEEE floating point standard:

Example: 22.625

Floating point form: 1.0110101_2 □ 2^4



Character Objects

Store numeric codes (ASCII and Unicode are standard

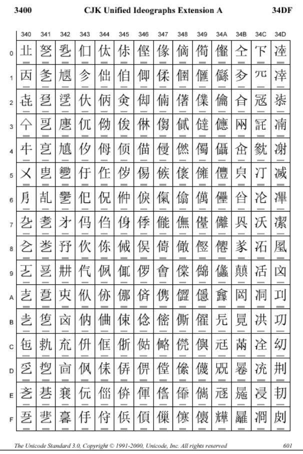
ASCII uses 1 byte (8 bits) per character, allowing for $2^8 = 255$ characters

$$\underbrace{\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}}_{c}$$

Java uses Unicode, which uses 2 bytes (16 bits) per character, allowing for $2^{16} = 65536$ characters (see examples on p. 68).

$$\underbrace{\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}}_{c} \quad \underbrace{\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}}_{\pi}$$

UNICODE supports a number of different character types (see www.unicode.org)



3400 CJK Unified Ideographs Extension A 34DF

	340	341	342	343	344	345	346	347	348	349	34A	34B	34C	34D
0	㐀	㐐	㐠	㐰	㑀	㑐	㑠	㑰	㒀	㒐	㒠	㒰	㓀	㓐
1	㐁	㐑	㐡	㐱	㑁	㑑	㑡	㑱	㒁	㒑	㒡	㒱	㓁	㓑
2	㐂	㐒	㐢	㐲	㑂	㑒	㑢	㑲	㒂	㒒	㒢	㒲	㓂	㓒
3	㐃	㐓	㐣	㐳	㑃	㑓	㑣	㑳	㒃	㒓	㒣	㒳	㓃	㓓
4	㐄	㐔	㐤	㐴	㑄	㑔	㑤	㑴	㒄	㒔	㒤	㒴	㓄	㓔
5	㐅	㐕	㐥	㐵	㑅	㑕	㑥	㑵	㒅	㒕	㒥	㒵	㓅	㓕
6	㐆	㐖	㐦	㐶	㑆	㑖	㑦	㑶	㒆	㒖	㒦	㒶	㓆	㓖
7	㐇	㐗	㐧	㐷	㑇	㑗	㑧	㑷	㒇	㒗	㒧	㒷	㓇	㓗
8	㐈	㐘	㐨	㐸	㑈	㑘	㑨	㑸	㒈	㒘	㒨	㒸	㓈	㓘
9	㐉	㐙	㐩	㐹	㑉	㑙	㑩	㑹	㒉	㒙	㒩	㒹	㓉	㓙
A	㐊	㐚	㐪	㐺	㑊	㑚	㑪	㑺	㒊	㒚	㒪	㒺	㓊	㓚
B	㐋	㐛	㐫	㐻	㑋	㑛	㑫	㑻	㒋	㒛	㒫	㒻	㓋	㓛
C	㐌	㐜	㐬	㐼	㑌	㑜	㑬	㑼	㒌	㒜	㒬	㒼	㓌	㓜
D	㐍	㐝	㐭	㐽	㑍	㑝	㑭	㑽	㒍	㒝	㒭	㒽	㓍	㓝
E	㐎	㐞	㐮	㐾	㑎	㑞	㑮	㑾	㒎	㒞	㒮	㒾	㓎	㓞
F	㐏	㐟	㐯	㐿	㑏	㑟	㑯	㑿	㒏	㒟	㒯	㒿	㓏	㓟

The Unicode Standard 3.0, Copyright © 1991-2000, Unicode, Inc. All rights reserved 601

Representing Booleans

- Only two possible values
  - **`true`** and **`false`**
- Only need two possible numbers, 0 and 1
- Single bit is all that is needed

Sec. 2.3

Some Basic Program Features

Using classes, we can build new types to model real world objects that cannot be represented using available types.

Pattern:

```
class ClassName extends ExistingClassName
{
   // attributes (variables & constants)
   // and behaviors (methods)
}
```

- ***`ClassName`*** is the name of a new reference type
- ***`ExistingClassName`*** is any class name known to the compiler
- { and } mark the boundaries of the declaration

An object is a program entity whose type is a class.

Importing Packages

Related classes can be grouped together into a container called a "package." A program specifies in what package to find a desired class

- *Fully-qualified name* of a class:
  `PackageName.ClassName`
  `PackageName1.PackageName2.ClassName`
- Using **`import`** ***`PackageName;`*** makes it possible to omit the prefixes and dot notation.
- Pattern:
  `import` ***`PackageName.*;`*** or
  `import PackageName.ClassName;`
  where ***`ClassName`*** is any class stored in ***`PackageName`***

Using Methods

- We *call*, *invoke*, or *send a message to* a method of an existing object, by using dot notation.

  Pattern:

  `objectName.methodName(arguments)`
- Example,

  `theScreen.print(" … ");`
  - **`theScreen`** is the object
  - **`print( )`** is the method being called

Sec. 2.4

Java Documentation – API

- Java designers have provided over 1600 classes
  - Called the Java Application Programmer's Interface or API
  - Each class provides variety of useful methods
  - Classes grouped into packages
- To find a needed package or class, use the hypertext-based documentation system:

http://java.sun.com/j2se/1.4.1/docs/api

This is an important reference source and you should learn to use it effectively