Lecture 07:
Data

Sierra College
CSCI-12
Spring 2015
Weds 02/18/15

Announcements

- **Schedule**
  - Some minor tweaks to schedule of assns, in RED
- **Past due assignments**
  - HW05: Numbers, accepted thru Fri 2/10 @ 11pm
- **Current assignments**
  - HW06: Template, due Weds 2/18 @ 11pm *(lab time today)*
- **New assignments**
  - HW07: Variables, due Tues 2/24 @ 11pm *(lab time today)*
    - Create some simple variables, and then display them back out
    - You DON'T need to do anything with this for this assn

Lecture Topics

- **Last time**:
  - Beginnings of the language
  - Basic structural elements of Java
- **Today**: all about DATA in Java
  - Variables
  - Datatypes
  - Literals, constants
  - Setting up data

Variables

- **Variables** are containers ("boxes") for data in a Java program
  - Named locations in memory for program data
  - We can refer to these values **symbolically** in our programs, rather than by some lengthy hex address
- Variables represent program data quantities which are likely to change, or vary, during the execution of a program
  - They store one data value at any point in time
  - But that value is subject to change over the course of program execution
- Variables typically represent one of three types of things:
  - Inputs, output, and internal calculations
  - The three general sections of code pseudocoded into your recent template's main() method
- Variables have the following properties:
  - **Name**
  - **Datatype**
  - **Value**

Variable Properties

- Variables have the following three properties:
  - A **name**
    - Tells where to find it in memory
    - Easier to use a symbolic name than a hex memory address
    - A variable name is one type of an **identifier** (user-specified name)
    - Subject to both identifier requirements and good naming conventions
  - A **datatype**
    - Tells us the fundamental nature of the contained data
    - Tells how big the data can be, determines allowable values
    - Tells compiler how much memory to allocate for it
    - There are 8 primitive datatypes
    - We'll later see that classes are effectively user-defined datatypes
  - A **value**
    - Tells what data is actually stored in that memory location:
    - Set as a result of initialization or assignment

```
int numStudents = 30;
```

0001 1110

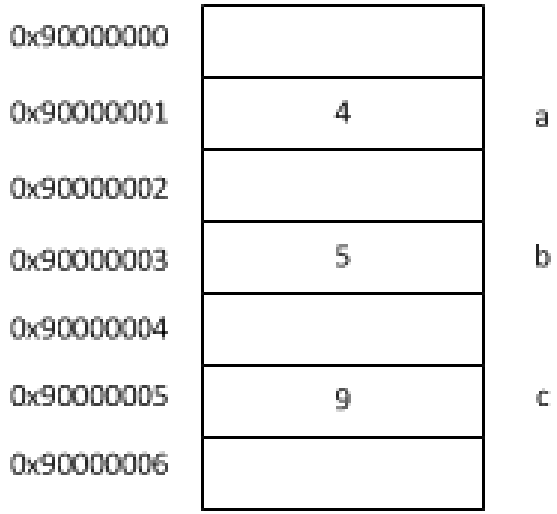
(0x1E)

Variables In Memory

Java code

```
39
40    public static void main(String [] args) {
41
42        // variable declarations
43        int a;
44        int b;
45        int c;
46
47        // variable initializations
48        a = 4;
49        b = 5;
50
51        // computations (variable assignment)
52        c = a + b;
53
54    }
55
```

Memory allocation



The compiler allocates space for declared variables “somewhere” in memory

(Our program doesn’t know, or care, exactly where the data actually gets stored)

Datatypes

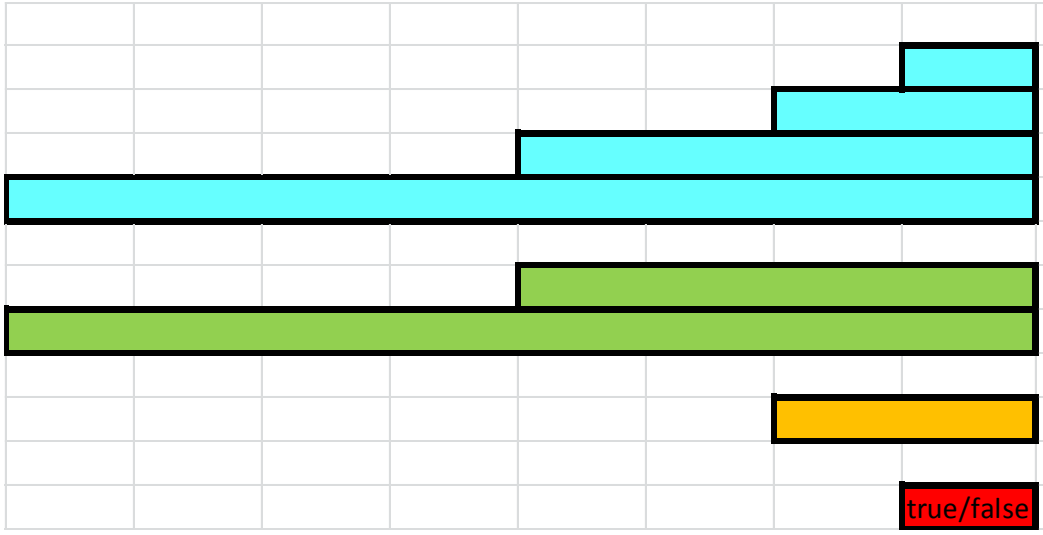
- If variables are “boxes”, then datatypes are the “box sizes”
  - What type of box?
  - What size of box?
  - What is the “nature” of the data that can be contained?
- **Datatypes** specify:
  - To the compiler:
    - How much memory to allocate for a variable
    - The format in which to store the data
  - To the developer:
    - The range of values which may be stored in a given variable
    - The “nature” of the data that can be stored in a given variable

Available Datatypes

- Java supports **8 primitive datatypes** as part of the core Java language:
  - Integer types: ***byte, short, int, long***
  - Floating-point types: ***float, double***
  - Character type: ***char***
  - True/false type: ***boolean***
- These are the basic atomic "matter" out of which ALL Java classes are ultimately created
  - Similar to the ~115 elements of the periodic table

Datatypes Are Just “Box Sizes”

- These variable sizes are to scale
- Nothing but boxes of differing sizes and types!



bytes	bits	datatype	notes
1	8	*byte*	
2	16	*short*	
4	32	*int*	
8	64	*long*	
4	32	*float*	
8	64	*double*	
2	16	*char*	
		boolean	T/F values only

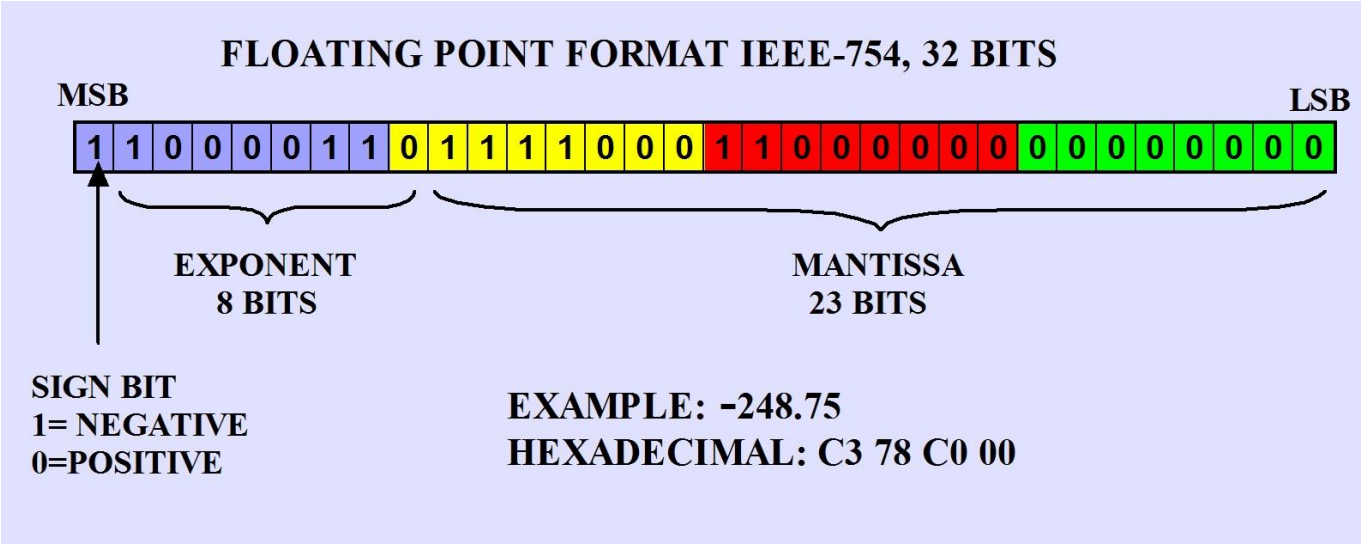
Datatype Sizes: Integer Types

- The **4 integer datatypes** are all signed types
  - The 4 integer datatypes all have different byte sizes
    - {1, 2, 4, 8} bytes
- With N bits, we can represent 2^N distinct values
  - Normally, N bits would allow a range from 0 → $(2^N - 1)$
  - If signed (+/-), min/max range is halved, but now shifted (centered) about 0
  - Signed ranges are now: -2^{N-1} → $+(2^{N-1} - 1)$
  - Example for N=4 bits:

N = 4:																													
	min	max																											
	0	15										0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		*16 distinct values*
	-8	7		-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7										*16 distinct values*

Datatype Sizes: Floating-Point Types

- The **2 floating-point datatypes** use IEEE-754 floating point formats for internal storage
  - **The 2 floating-point datatypes are 4 and 8 bytes in size**
  - Tradeoffs between **range** and **precision**
  - Only so many bits available for mantissa, exponent, and sign bit



Datatype Sizes: Other Types

- The ***char* datatype** is 2 bytes
  - Handles Unicode characters from 0x0000-0xFFFF (0-65,535)
  - Unicode is an international standard which maps symbols in all human languages to specific hex character codes
    - See http://unicode-table.com/en/ , link is in Canvas module
  - The first 128 characters of Unicode are synonymous with the ASCII character table

- The ***boolean* datatype** handles only 2 binary values
  - ***true*** and ***false*** (Java reserved word values)

Integer Datatypes

Integer Datatype	Size In Bytes	Minimum Value	Maximum Value
byte	1	-128	127
short	2	-32,768 (~ -32K)	32,767 (~ +32K)
int	4	-2,147,483,648 (~ -2.1B)	2,147,483,647 (~ +2.1B)
long	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

- **Use integer datatypes for**:
  - Whole-valued numbers
  - Numbers not requiring any decimal precision
- **Examples**:
  - Counts, quantities, populations, ages, ID numbers, …
- In THIS COURSE, we typically will not use the ***byte*** and ***short*** types

Example declarations:

```
int     testGrade;
int     numPlayers, highScore, diceRoll;
short   xCoordinate, yCoordinate;
byte    ageInYears;
long    cityPopulation;
```

Floating Point Datatypes

Floating Point Datatype	Size In Bytes	Minimum Value	Resolution	Maximum Value
float	4	-3.40E+38	1.40E-45	+3.40E+38
double	8	-1.80E+308	4.94E-324	1.80E+308

- **Use floating point datatypes for**:
  - Numbers with fractional parts
  - Numbers requiring decimal accuracy
  - Precise measurements
  - Very large or small numbers requiring scientific notation
- **Examples**:
  - Costs, currencies, averages, mathematical or physical calculations, etc.

Example declarations:

```
float    salesTax;
double   interestRate;
double   paycheck,  sumSalaries;
float    battingAverage;
double   netForceMeasured;
double   sineOfAngle;
```

Character Datatype

- The ***char*** datatype represents one 16-bit (2 byte) Unicode character
- Can be specified as a single character OR as a number (decimal or hex)

Character Datatype	Size In Bytes	Minimum Value	Maximum Value
char	2	Unicode 0x0000 (null character)	Unicode 0xFFFF ("not a character")

- **Use the *char* datatype for**:
  - Single characters within SINGLE quotes
- **Examples**:
  - Letter grades
  - User options
  - Keyboard input characters
  - Special characters or symbols

Example declarations:

```
char    finalGrade, gender;
char    userInputOption, keyPressed;
char    newline, tab, doubleQuote;
char    yenSymbol, euroSymbol, poundSymbol;
```

Boolean Datatype

- The ***boolean*** datatype accepts two specific values ONLY

Boolean datatype	Size In Bytes	Value #1	Value #2
boolean	Unspecified: JVM-specific	***true***	***false***

- **Use the *boolean* datatype for**:
  - Binary T/F data or settings
  - Decision making states
- **Examples**:
  - Looping or branching flags
  - Yes/no states
  - On/off or high/low settings

Example declarations:

```
boolean  passedClass;
boolean  checkEngineLight;
boolean  continueLooping;
boolean  readMoreData;
```

Object Datatypes

- Class names in Java also serve as “datatypes”
  - Classes provided by Java (“extensions” of the core language)
  - Classes that we create (“user-defined datatypes”)
- When we **instantiate** (create) objects from classes, this tells the compiler how much memory to allocate for that object
- From 2nd lab Hello Again:

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

    // declare variable
    HelloRL person;

    // initialize variable
    person = new HelloRL("Rob", "Lapkass");

    // print message
    person.printGreeting();
}
```

The ***String*** Class

- The predefined Java class ***String*** is commonly used for **text** data
  - **Text** → any characters within DOUBLE quotes: "I am some text"

    String firstName

    String lastName;

    String guiPromptString;
  - All of the above are Java **objects** of type *String*
- ***String*** **is a Java class**, and <u>not</u> a primitive datatype
  - But, it is often mistaken for one
  - It sure seems like it should be the 9th primitive datatype!
- This is a very common example of using other Java classes as additional, special-purpose datatypes

Variable Declaration

- Variables must be **declared** prior to first usage
  - Declaration tells the compiler how much memory to allocate, and format to use in storing the data
  - If we try to use an undeclared variable, it results in a **compiler error**
- General syntax form(s):

```
datatype  variable;                  // one per statement
datatype  variable1, variable2, …    // multiple in one statement
```

- Variable declaration is typically done at the beginning of any class or method
  - This is **good programming practice**
    - Avoid creating new variables all over your code
  - **We will also adhere to this a course coding standard**
- We can declare a variable without setting a value for it (yet)

Variable Declaration Examples

```
20          // variable or constant declaration alone
21          byte age;
22          short xCursor;
23          int testScore;
24          long cityPopulation;
25          float salesTaxRate;
26          double yearlySalary;
27          char gender;
28          boolean isEmpty;
```

- See ***VariableExamples.java*** in Example Source Code

Variable Initialization

- Before they can be used, variables must also be **initialized**
  - **Initialization** assigns a starting value to a variable
  - An uninitialized variable will also result in a compiler error
- General syntax form(s):

```
variable = value;          // one initialization
variable2 = variable1;     // variable1 must already be initialized
variable = expression;     // a valid, type-compatible Java statement
```

- Variable initialization may be done:
  - Near the beginning of a class or method
  - During the course of program execution

Variable Initialization Examples

```
30          // variable or constant initialization alone
31          age = 21;
32          xCursor = 45;
33          testScore = 98;
34          cityPopulation = 125000L;
35          salesTaxRate = 0.075F;
36          yearlySalary = 89400.0;
37          gender = 'M';
38          isEmpty = false;
```

- Note the special termination characters for ***float*** and ***long*** types
  - We'll cover this some more when we get to **literals**…
- See ***VariableExamples.java*** in Example Source Code

Variable Declaration + Initialization

- Variable **declaration** and **initialization** may be combined into one single statement
- General syntax form(s):

  datatype variable = value;

  datatype variable2 = variable1; // variable1 must already exist

  datatype variable = expression; // a valid, type-compatible Java statement

- Variable declaration + initialization is typically done at the beginning of any class or method
  - If we already know the starting value, just initialize to that value

Variable Declaration + Initialization Examples

```
40          // variable or constant declaration + initialization
41          byte age = 21;
42          short xCursor = 45;
43          int testScore = 98;
44          long cityPopulation = 125000L;
45          float salesTaxRate = 0.075F;
46          double yearlySalary = 89400.0;
47          char gender = 'M';
48          boolean isEmpty = false;
```

- Note the special termination characters for ***float*** and ***long*** types
  - We'll cover this some more when we get to literals…
- See ***VariableExamples.java*** in Example Source Code

Variable Assignment

- Once a variable is declared, it may have its value **assigned** (changed)
  - Upon initialization
  - Together in one step (declaration + initialization)
  - At any time during subsequent program execution
- Variable assignment uses the **assignment operator**: **=**
  - Right-to-left evaluation
  - Evaluate right side **expression** fully, then assign that single value to the left side variable
- The right side may consist of:
  - Some literal value
  - Another variable
  - Any valid **expression** (variables/operators/constants)

Variable Assignment Examples

- x = 20;
- y = x;
- z = x + y + 50;

- x gets the literal value 20
- Now y gets the value of x (20)
- Evaluate the right side, then assign it to the left side for z (90)

- int age = 20;
- age = sam.getAge();

- Declare AND initialize age
- Overwrite age using an object method

- double a=4.0, b=6.0;

- Multiple declarations and initializations in one statement

Variable Naming Conventions

- Variables MUST adhere to the rules of Java identifiers
  - Numbers and letters, no spaces, no keywords, etc.
- Variables SHOULD follow good **variable naming conventions:**
  - Variable names should be "descriptive", and should clearly describe what is contained in them
  - Variable names should be neither too terse, nor too lengthy
  - Variable names should represent "things" (nouns)
  - Variable names should begin with a lower case letter
  - Variable names should be in "camelCase", with words after the first beginning with a capital letter
  - Avoid using underscores and the $ sign in variable names
- Following such conventions leads to **self-documenting code**
  - Even if you are unfamiliar with a piece of code, well-formed variable names strongly suggest at their purpose

Java Variable Name Examples

Allowable, but we can do better	Good Java variable names
x	xCursor
str	strUserPrompt
name	firstName, lastName
bankaddress	bankAddress
class_avg_grade	classAvgGrade
Joe_test_score	joeTestScore
dolamt	dollarAmt
somereallyreallylongvar	longVarName

Variable Name Exceptions

- Of course, there are always reasonable exceptions
- Sometimes we DO use shorthand variable names
  - For temporary, trivial, “throwaway” variables
  - When the meaning is clear from the context
- Some examples:
  - Characters: c, ch
  - Loop counters: i, j, k
  - Coordinates: x, y, z
  - Exceptions: e
  - Strings: str, s
  - Streams: in, out
  - Graphics: g
- These could be the variable names themselves, or perhaps standard prefixes to the variable names

Strong Datatyping

- Java is a **strongly-typed language**
  - Variables must be declared as specific datatypes upfront
  - Compiler strictly monitors statements for compatible typing
- Mixed type expressions are permitted, though
  - You can assign a smaller datatype variable to a larger datatype variable
  - You CANNOT assign a larger datatype variable to smaller datatype variable

Compatible Data Types

A variable of any type in right column can be assigned to a variable of any type in left column:

Data Type	**Compatible Data Types**
byte	*byte*
short	*byte, short*
int	*byte, short, int, char*
long	*byte, short, int, long, char*
float	*float, byte, short, int, long, char*
double	*float, double, byte, short, int, long, char*
boolean	*boolean*
char	*char*

Literals

- **Literals** are numeric or textual values used to assign specific data to variables
  - Their values are “literally” whatever is shown
  - They represent “hardwired” values in a program
- There are some nuances to some of the datatypes

- See **LiteralExamples.java** in Example Source Code
  - Code for the following few slides

Integer Literals

- For *int*, *short*, *byte*, *long*:
  - Optional leading +/- sign
  - Remainder is digits 0-9 or A-F, in any combination
  - A beginning 0x indicates a hex integer
- For *long* only:
  - Terminates with an 'l' or a 'L' (preferred) ← a common "gotcha"
- Examples:

```
int absoluteZero = -273;
long cityPopulation = 4235873L;
short opCode = 0xDE;
```

See **LiteralExamples.java** in Example Source Code

Floating-Point Literals

- For *float*:
  - Optional leading +/- sign
  - Remainder in fixed-point or scientific format
  - Terminated with an 'F' or 'f' ← a common "gotcha"
- For *double*:
  - Optional leading +/- sign
  - Remainder in fixed-point or scientific format
- Examples:

```
float salesTaxRate = 0.0875F;
double speedOfLight = 3.0E8;
```

See **LiteralExamples.java** in Example Source Code

char Literals

- For *char*:
  - Any printable character in single quotes
  - A decimal value from 0 – 65535, or hex value 0x0000 – 0xFFFF
  - An escape sequence '***x**':
    - \n newline
    - \t tab
    - \" embedded double quote
    - \' embedded single quote
    - \u represents a Unicode character
- Examples:

```
char gender = 'F';
char tabChar = 0x09;
char newline = '\n';
```

See **LiteralExamples.java** in Example Source Code

boolean Literals

- Boolean literals are simple: there are only two possible values
  - ***true***
  - ***false***
- No single or double quotes required, these are Java reserved words
- Examples:

```
boolean isPassing = true;
boolean heaterOn = false;
```

See **LiteralExamples.java** in Example Source Code

String Literals

- *String* is a Java class, not a primitive datatype
  - String variables are really **objects**
- A *String* literal is any text enclosed within double quotes
  - The string concatenation operator '+' may also be used
  - Escape sequences may be inlined within a string literal
  - Variable values may also be combined with string literals
- Examples:

```
System.out.println("Hello  " + "World!");
System.out.println("I want this\nPrinted on two lines");
System.out.println("age = \t\t" + age);
```

See **LiteralExamples.java** in Example Source Code

Unicode Example

```
char euro1 = '\u20AC';          // Euro symbol as a Unicode variable literal
char euro2 = 0x20AC;            // Euro symbol as a Unicode variable hex constant

System.out.println("as a char variable literal: \t\t" + euro1);
System.out.println("as a char variable hex constant: \t" + euro2);
System.out.println("in a string literal: \t\t\t\u20AC");
```

```
----jGRASP exec: java Unicode

as a char variable literal:             €
as a char variable hex constant:        €
in a string literal:                    €

----jGRASP: operation complete.
```

- 3 ways to display the same Unicode character
- Some tabs “tweaking” required for nice output alignment

- See **Unicode.java** in Source Code Examples

Constants

- **Constants** are simply regular variables which have some "special identification"
  - They remain truly unchanged during execution
  - They warrant some distinct, recognizable naming
- Good practice may dictate a special constant designation, to prevent any value changes by your code
  - Use the ***final*** keyword to prevent any value changes
- General syntax:

  ***final* datatype CONSTANT_IDENTIFIER = constantValue;**
- Using constants helps us avoid having to maintain lots of hardwired numeric literals sprinkled throughout our code
- Some common usages:
  - Physical constants
  - Conversion factors: between units, between currencies
  - Character or *String* constants

Constant Conventions/Examples

- By good programming convention, constants:
  - Use the ***final*** keyword in their declaration
  - Are named using all CAPITALS
  - Have underscores between words

- Examples:

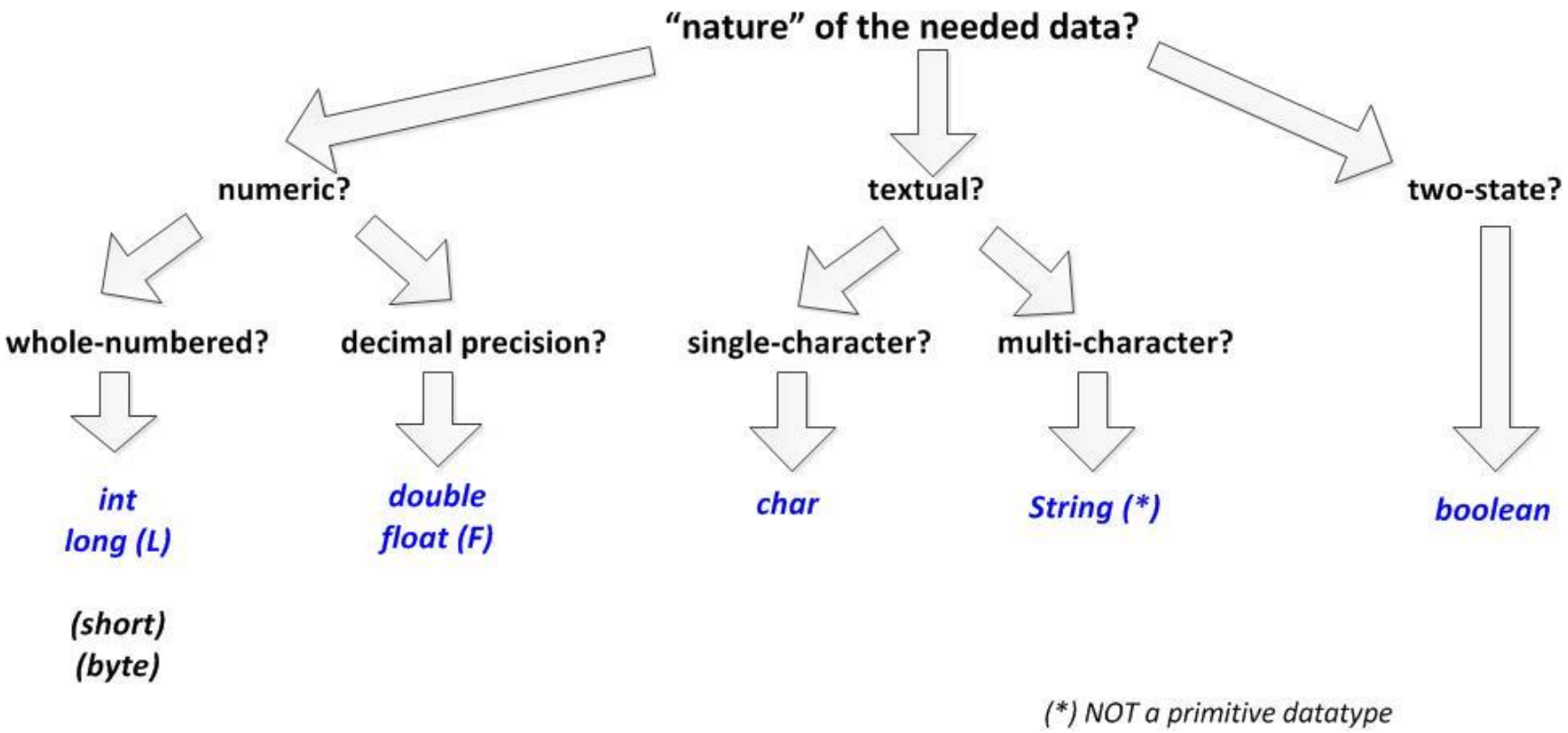
```
final int CARDS_IN_DECK = 52;
final int ONE_DOZEN = 12;
final long ONE_MILLION = 1000000L;
final float PLACER_SALES_TAX_RATE = 0.075F;
final double METERS_TO_INCHES = 39.37;
final double INCHES_TO_METERS = 1/METERS_TO_INCHES;  // reciprocal
final double EARTH_GRAVITY = -9.81;     //  [m/s^2]
final char YEN_SYMBOL = 0x00A5;
final String PROMPT_TEXT = "Enter an input value:  ";
```

- See **ConstantExamples.java** in Example Source Code

Datatype Considerations

- Start by examining the data needed by the problem you are trying to solve
- In general:
  - Is your data inherently numerical, textual, or boolean?
  - Do you require whole-number or decimal resolution?
    - Discrete numbers of things, counts → integer types
    - Measurements, money, precision, etc. → floating point types
  - What is the expected range of values? (min/max)
  - Select the smallest datatype which will hold expected data
- But more practically and realistically:
  - For integer data, just use ***int*** unless a ***long*** is called for
  - For floating point data, use either a ***double*** or ***float***
  - Sometime, the interface of pre-existing Java classes will dictate the datatypes to be used (such as ***long*** or ***double***)

Datatype Flowchart



Data Definition Algorithm

- Start with a **clear understanding or description** of the problem at hand
- What are all the **data quantities** needed?
  - Are they inputs? outputs? calculated values?
- What is the fundamental **NATURE** of each piece of data?
  - Select appropriate **datatypes** for each piece of data
  - Use preceding decision tree
- Assign suitably descriptive **variable names**
- Does the value change or remain constant?
  - Does the variable's purpose warrant promotion to a **CONSTANT_NAME**?
- Do you know the value for the data?
  - If yes, declare and initialize the variable
  - If no, just declare the variable
    - Provide a reasonable starting "placeholder" for its value?
    - Calculate its value later in the program?

Data Definition Examples

- Population of a city
  - Inherently a count → use ***int***
- Bank account balances
  - Need 2-decimal precision → use ***float*** or ***double***
- Product code, database item codes
  - Lengthy unique integers → use ***int*** (maybe a ***long***?)
- High-precision weight, height, distances
  - Need decimal accuracy → use ***float*** or ***double***
- Mathematical or trig functions
  - Depends upon Java-provided methods → often ***doubles***
- Flag values, settings, states → use ***boolean***
- Single-character options, keystrokes → use ***char***

For Next Time

- **Lecture Prep**
  - Text readings and lecture notes
- **Assignments**
  - See slide 2 for new/current/past due assignments