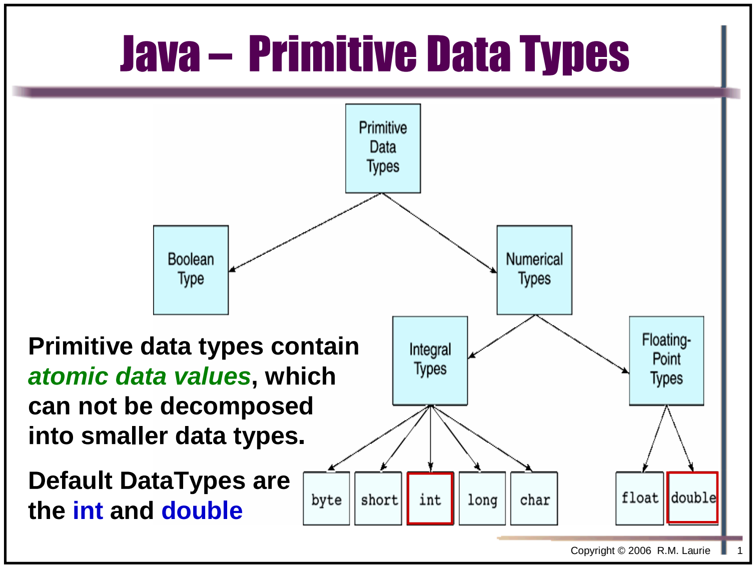
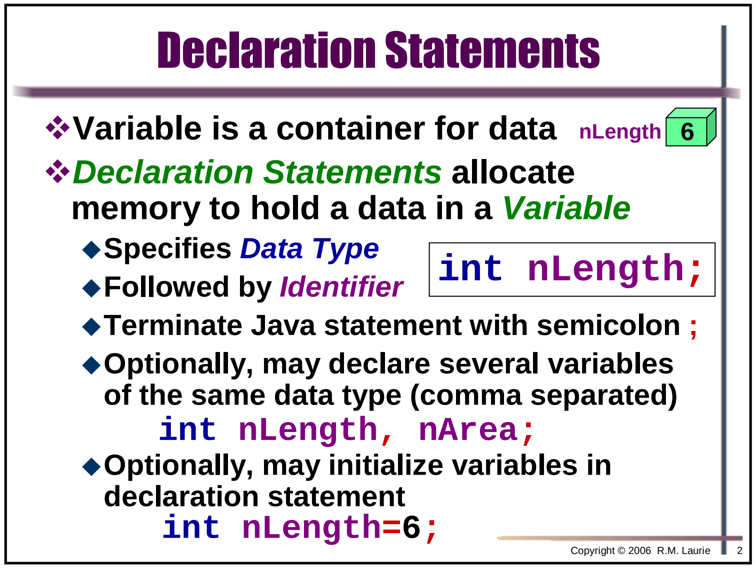


Java – Primitive Data Types

Primitive Data Types
Boolean Type
Numerical Types
Integral Types
Floating-Point Types
byte
short
int
long
char
float
double

Primitive data types contain *atomic data values*, which can not be decomposed into smaller data types.

Default DataTypes are the int and double



Declaration Statements

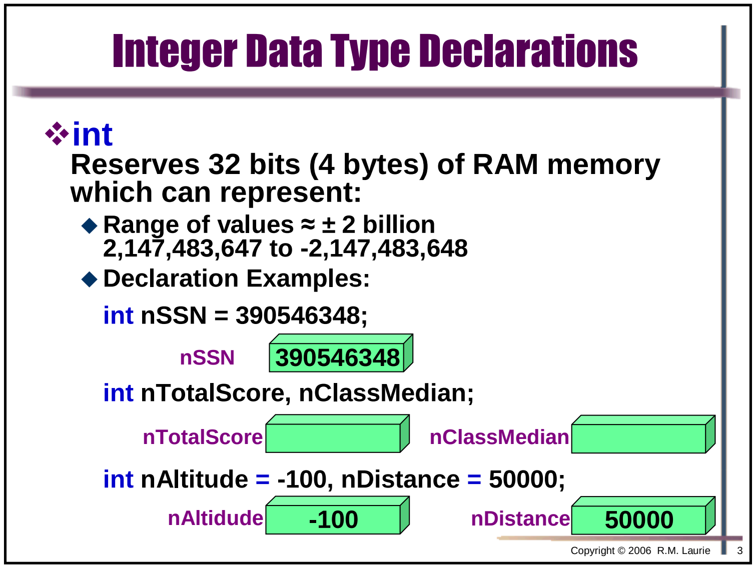
- Variable is a container for data nLength 6
- *Declaration Statements* allocate memory to hold a data in a *Variable*
  - Specifies *Data Type*
  - Followed by *Identifier*
  - Terminate Java statement with semicolon ;
  - Optionally, may declare several variables of the same data type (comma separated)

```
int nLength;
```

```
int nLength, nArea;
```

  - Optionally, may initialize variables in declaration statement

```
int nLength=6;
```



Integer Data Type Declarations

- int
  Reserves 32 bits (4 bytes) of RAM memory which can represent:
  - Range of values ≈ ± 2 billion
    2,147,483,647 to -2,147,483,648
  - Declaration Examples:

int nSSN = 390546348;

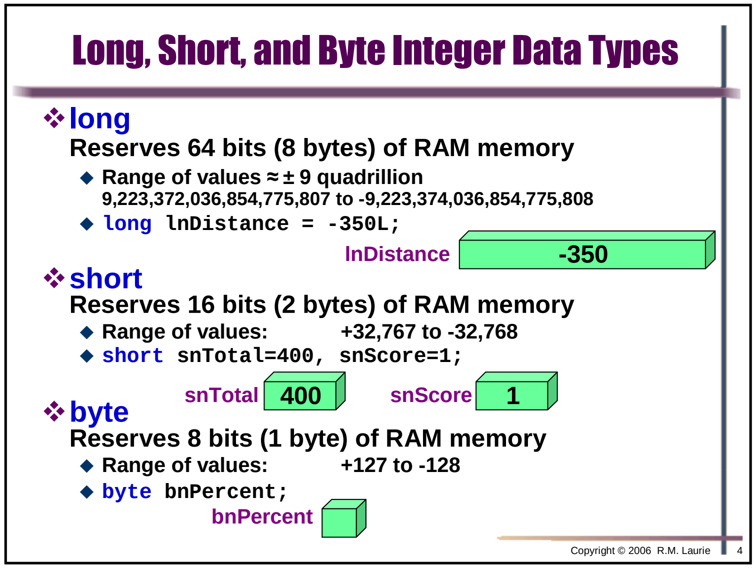
nSSN 390546348

int nTotalScore, nClassMedian;

nTotalScore nClassMedian

int nAltitude = -100, nDistance = 50000;

nAltidude -100 nDistance 50000



Long, Short, and Byte Integer Data Types

- long
  Reserves 64 bits (8 bytes) of RAM memory
  - Range of values ≈ ± 9 quadrillion
    9,223,372,036,854,775,807 to -9,223,374,036,854,775,808
  - `long lnDistance = -350L;`

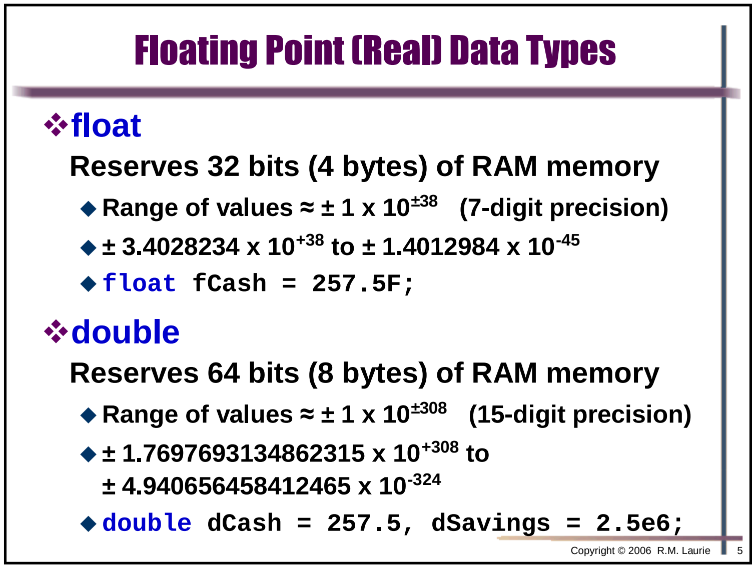
lnDistance -350

- short
  Reserves 16 bits (2 bytes) of RAM memory
  - Range of values: +32,767 to -32,768
  - `short snTotal=400, snScore=1;`

snTotal 400 snScore 1

- byte
  Reserves 8 bits (1 byte) of RAM memory
  - Range of values: +127 to -128
  - `byte bnPercent;`

bnPercent

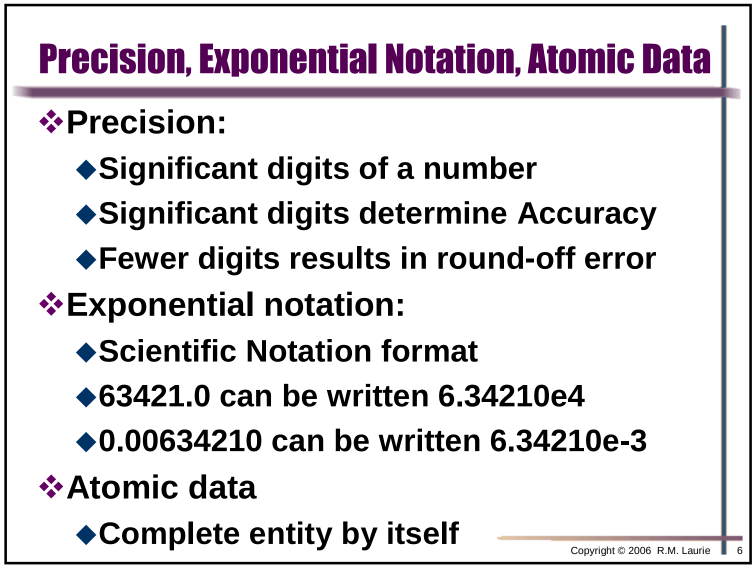


Floating Point (Real) Data Types

- **float**

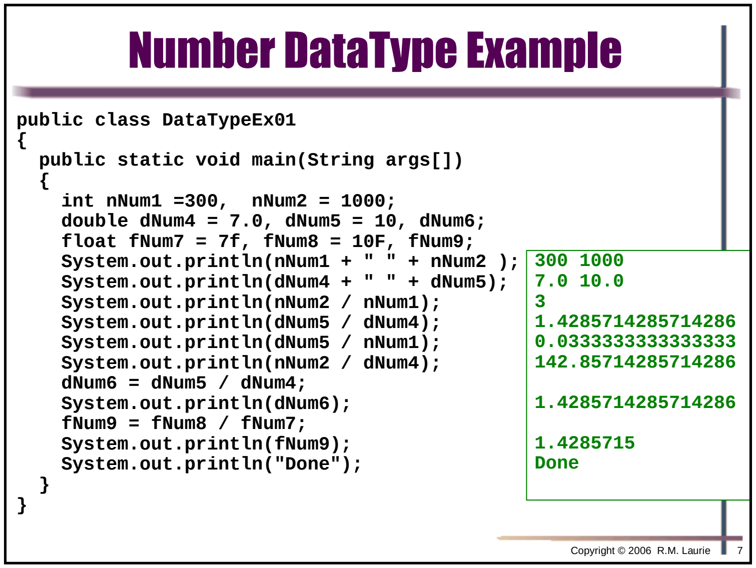
  Reserves 32 bits (4 bytes) of RAM memory
  - Range of values ≈ $\pm 1 \times 10^{\pm 38}$ (7-digit precision)
  - $\pm 3.4028234 \times 10^{+38}$ to $\pm 1.4012984 \times 10^{-45}$
  - `float fCash = 257.5F;`
- **double**

  Reserves 64 bits (8 bytes) of RAM memory
  - Range of values ≈ $\pm 1 \times 10^{\pm 308}$ (15-digit precision)
  - $\pm 1.7697693134862315 \times 10^{+308}$ to $\pm 4.940656458412465 \times 10^{-324}$
  - `double dCash = 257.5, dSavings = 2.5e6;`



Precision, Exponential Notation, Atomic Data

- Precision:
  - Significant digits of a number
  - Significant digits determine Accuracy
  - Fewer digits results in round-off error
- Exponential notation:
  - Scientific Notation format
  - 63421.0 can be written 6.34210e4
  - 0.00634210 can be written 6.34210e-3
- Atomic data
  - Complete entity by itself



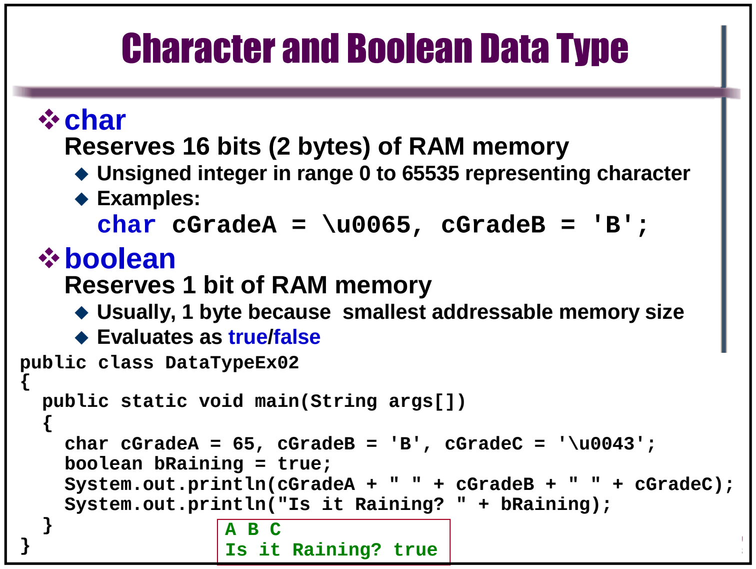
Number DataType Example

```
public class DataTypeEx01
{
  public static void main(String args[])
  {
    int nNum1 =300,  nNum2 = 1000;
    double dNum4 = 7.0, dNum5 = 10, dNum6;
    float fNum7 = 7f, fNum8 = 10F, fNum9;
    System.out.println(nNum1 + " " + nNum2 );
    System.out.println(dNum4 + " " + dNum5);
    System.out.println(nNum2 / nNum1);
    System.out.println(dNum5 / dNum4);
    System.out.println(dNum5 / nNum1);
    System.out.println(nNum2 / dNum4);
    dNum6 = dNum5 / dNum4;
    System.out.println(dNum6);
    fNum9 = fNum8 / fNum7;
    System.out.println(fNum9);
    System.out.println("Done");
  }
}
```

```
300 1000
7.0 10.0
3
1.4285714285714286
0.0333333333333333
142.85714285714286

1.4285714285714286

1.4285715
Done
```



Character and Boolean Data Type

- **char**

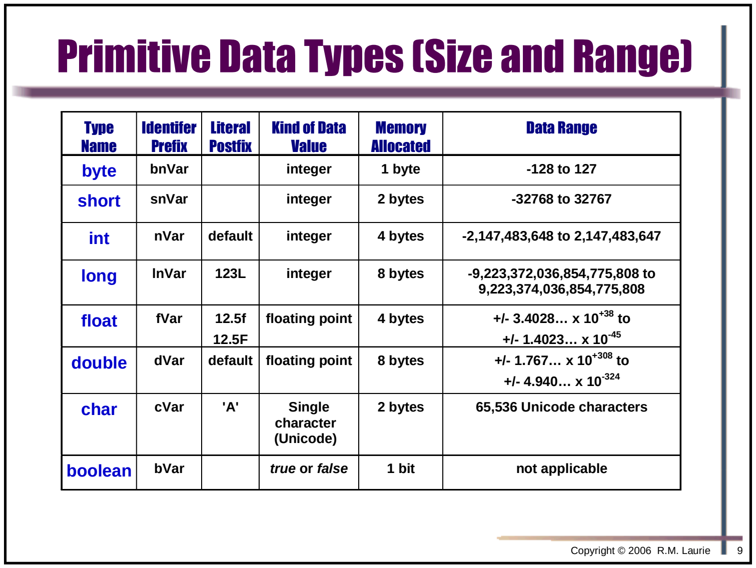
  Reserves 16 bits (2 bytes) of RAM memory
  - Unsigned integer in range 0 to 65535 representing character
  - Examples:

    `char cGradeA = \u0065, cGradeB = 'B';`
- **boolean**

  Reserves 1 bit of RAM memory
  - Usually, 1 byte because smallest addressable memory size
  - Evaluates as true/false

```
public class DataTypeEx02
{
  public static void main(String args[])
  {
    char cGradeA = 65, cGradeB = 'B', cGradeC = '\u0043';
    boolean bRaining = true;
    System.out.println(cGradeA + " " + cGradeB + " " + cGradeC);
    System.out.println("Is it Raining? " + bRaining);
  }
}
```

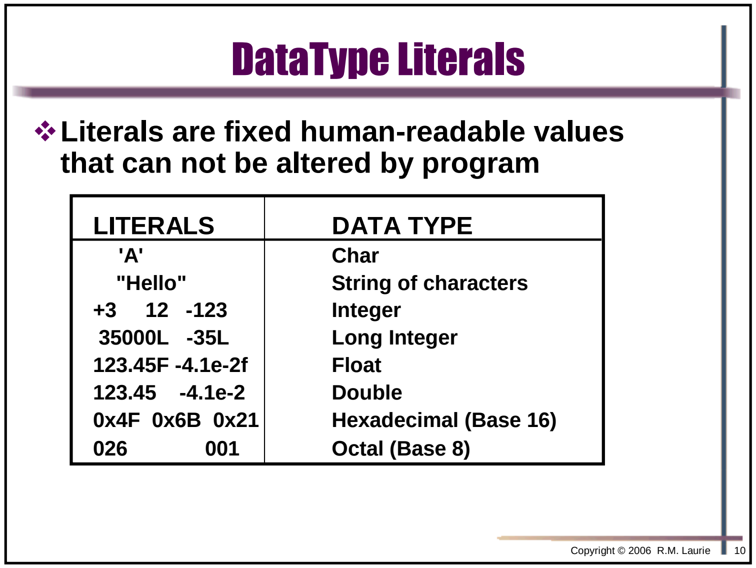
```
A B C
Is it Raining? true
```



Primitive Data Types (Size and Range)

Type Name	Identifer Prefix	Literal Postfix	Kind of Data Value	Memory Allocated	Data Range
byte	bnVar		integer	1 byte	-128 to 127
short	snVar		integer	2 bytes	-32768 to 32767
int	nVar	default	integer	4 bytes	-2,147,483,648 to 2,147,483,647
long	lnVar	123L	integer	8 bytes	-9,223,372,036,854,775,808 to 9,223,374,036,854,775,808
float	fVar	12.5f 12.5F	floating point	4 bytes	+/- 3.4028… x 10^{+38} to +/- 1.4023… x 10^{-45}
double	dVar	default	floating point	8 bytes	+/- 1.767… x 10^{+308} to +/- 4.940… x 10^{-324}
char	cVar	'A'	Single character (Unicode)	2 bytes	65,536 Unicode characters
boolean	bVar		*true* or *false*	1 bit	not applicable

9

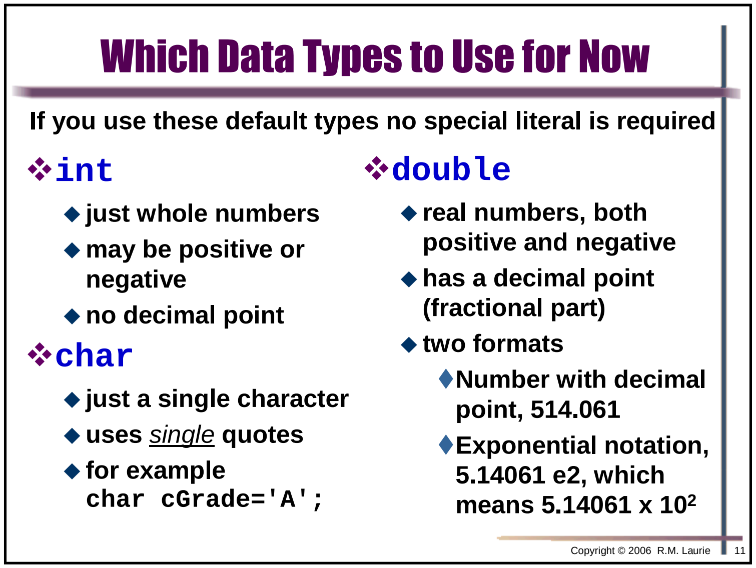


DataType Literals

- ❖ Literals are fixed human-readable values that can not be altered by program

LITERALS	DATA TYPE
'A'	Char
"Hello"	String of characters
+3 12 -123	Integer
35000L -35L	Long Integer
123.45F -4.1e-2f	Float
123.45 -4.1e-2	Double
0x4F 0x6B 0x21	Hexadecimal (Base 16)
026 001	Octal (Base 8)

10

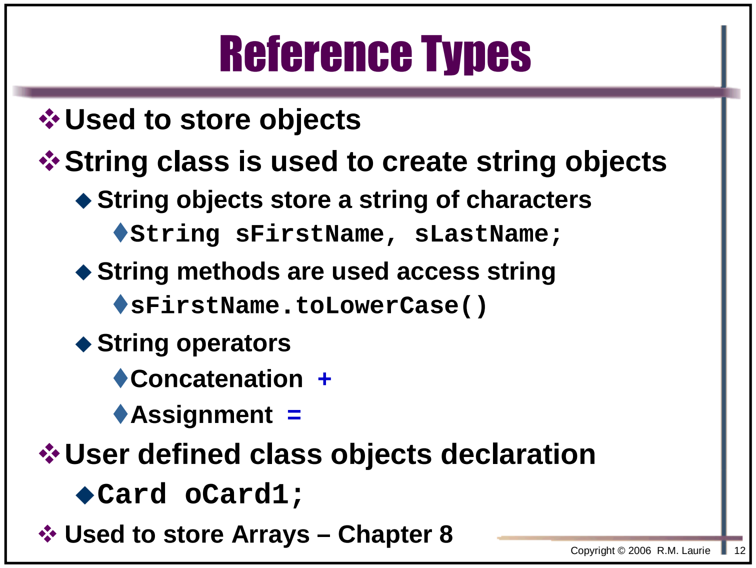


Which Data Types to Use for Now

If you use these default types no special literal is required

- ❖ `int`
  - ◆ just whole numbers
  - ◆ may be positive or negative
  - ◆ no decimal point
- ❖ `char`
  - ◆ just a single character
  - ◆ uses <u>single</u> quotes
  - ◆ for example `char cGrade='A';`
- ❖ `double`
  - ◆ real numbers, both positive and negative
  - ◆ has a decimal point (fractional part)
  - ◆ two formats
    - ♦ Number with decimal point, 514.061
    - ♦ Exponential notation, 5.14061 e2, which means 5.14061 x 10^2

11



Reference Types

- ❖ Used to store objects
- ❖ String class is used to create string objects
  - ◆ String objects store a string of characters
    - ♦ `String sFirstName, sLastName;`
  - ◆ String methods are used access string
    - ♦ `sFirstName.toLowerCase()`
  - ◆ String operators
    - ♦ Concatenation `+`
    - ♦ Assignment `=`
- ❖ User defined class objects declaration
  - ◆ `Card oCard1;`
- ❖ Used to store Arrays – Chapter 8

12

Construct a Data Declaration Section

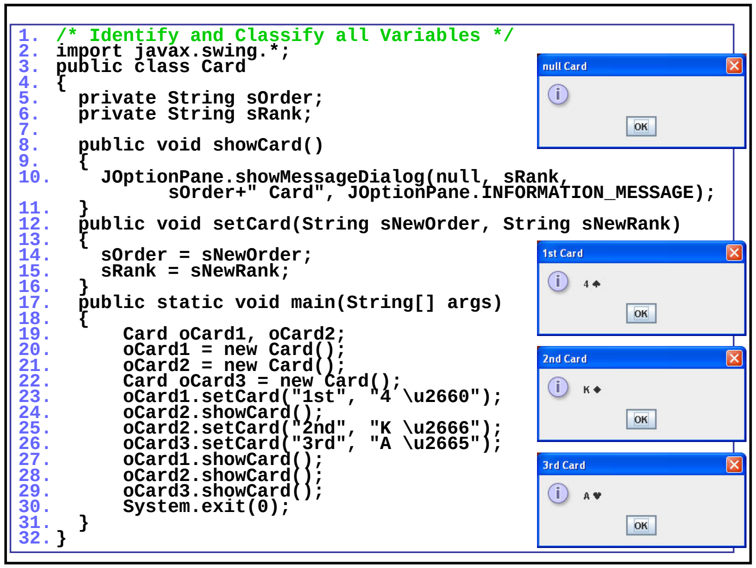
- Dependent on:
  - Variable placement within class
  - Presence or absence of reserved word `static`
- Classifications of variables:
  - *Local* – Within methods and used to create objects
    - Neither an access modifier or static are permitted
  - *Instance* – Within class's body but outside method
    - Every object gets variable of this type, static not permitted
  - *Class* – Within class's body but outside method
    - Part of class but not object, must use static keyword
  - *Parameter* – Within parenthesis of method header
    - Neither an access modifier or static are permitted
    - Used to pass data values to a method

Declaration Statements Syntax

- `optAccessSpecifier dataType varName;`
  - private – Access variables only within class methods
  - public – Access variables from anywhere (Avoid!)
- *Local* – Allocated only when method is executed
  - int nSum;
- *Instance* – Created for each object (Object Data)
  - private int nSum; // Access only within class methods
- *Class* – Within class's body but outside method
  - private static int nSum;
- *Parameter* – Within parenthesis of method head
  - public void setCard(String sOrder, int nRank)

Creating Objects

- Objects
  - Contains the *Instance Variables* declared in data declaration section
- Reference variable
  - Reference location for actual object's values
  - `Card oCard1;`
- `new` Dynamic Memory Allocation operator
  - For *creating an instance* or *instantiating an object*
  - `oCard1 = new Card();`
  - `Card oCard1 = new Card();` // Combined Form
- Methods
  - Provide operations that can be applied to objects
  - Object independent general-purpose functions



```java
/* Identify and Classify all Variables */
import javax.swing.*;
public class Card
{
  private String sOrder;
  private String sRank;

  public void showCard()
  {
    JOptionPane.showMessageDialog(null, sRank,
          sOrder+" Card", JOptionPane.INFORMATION_MESSAGE);
  }
  public void setCard(String sNewOrder, String sNewRank)
  {
    sOrder = sNewOrder;
    sRank = sNewRank;
  }
  public static void main(String[] args)
  {
      Card oCard1, oCard2;
      oCard1 = new Card();
      oCard2 = new Card();
      Card oCard3 = new Card();
      oCard1.setCard("1st", "4 \u2660");
      oCard2.showCard();
      oCard2.setCard("2nd", "K \u2666");
      oCard3.setCard("3rd", "A \u2665");
      oCard1.showCard();
      oCard2.showCard();
      oCard3.showCard();
      System.exit(0);
  }
}
```

null Card

1st Card

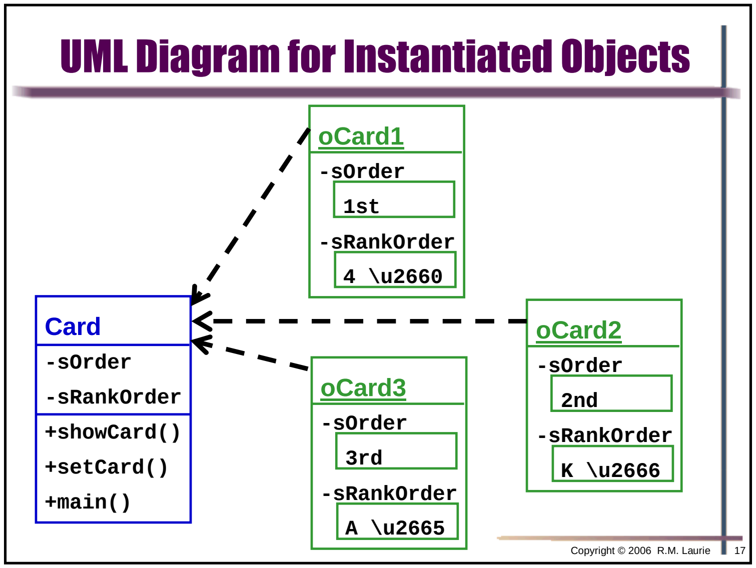
4 ♠

2nd Card

K ♦

3rd Card

A ♥



UML Diagram for Instantiated Objects

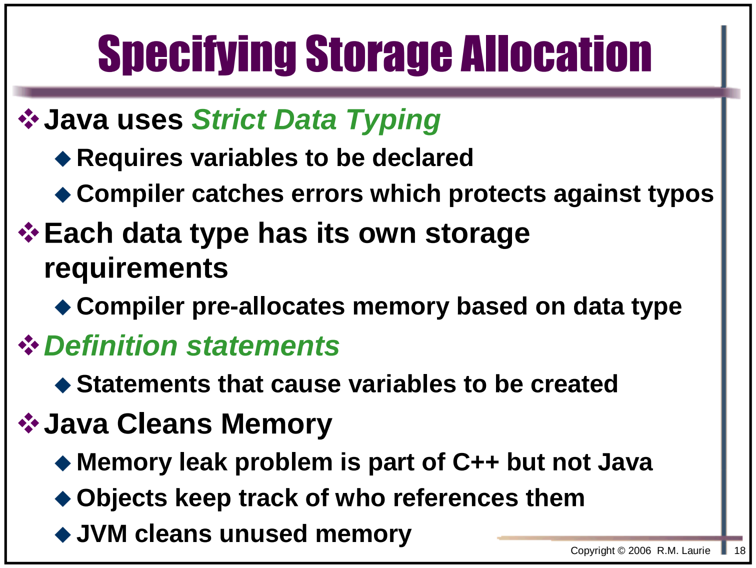
Card
-sOrder
-sRankOrder
+showCard()
+setCard()
+main()

oCard1
-sOrder
1st
-sRankOrder
4 \u2660

oCard2
-sOrder
2nd
-sRankOrder
K \u2666

oCard3
-sOrder
3rd
-sRankOrder
A \u2665

Copyright © 2006 R.M. Laurie 17



Specifying Storage Allocation

- Java uses *Strict Data Typing*
  - Requires variables to be declared
  - Compiler catches errors which protects against typos
- Each data type has its own storage requirements
  - Compiler pre-allocates memory based on data type
- *Definition statements*
  - Statements that cause variables to be created
- Java Cleans Memory
  - Memory leak problem is part of C++ but not Java
  - Objects keep track of who references them
  - JVM cleans unused memory

Copyright © 2006 R.M. Laurie 18