INFO 5100

Application Engineering Design

Java Data Types

Daniel Peters

d.peters@neu.edu

- Lecture
  1. Java Language Basics
  2. Java Data Types
  3. Java Primitive Types
  4. Java String class
  5. Java Reference Type
  6. Java Parameter Passing

Java Language

- Object Oriented Programming Language
  - Data
    - Memory used by the program
  - Program statements
    - Code instructing the actions of the processor
  - Class
    - Data
    - Methods (program code) operating on class data

Java Language

- Everything is a class
  - Definable aggregate containing data and methods
  - All data and code in Java exists only in context of a class
- Java Language Usage
  - Use class statically
  - Use object instantiated (created) from a class

Java Language

- Statically typed programming language

  "*The Java programming language is statically-typed, which means that all variables must first be declared before they can be used.*"

  – All data must be **declared** and made known to compiler before its first use

    DataType name;

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Java Language

- Statically typed Languages include:
  - Java
  - C
  - C++
- Dynamically typed languages include:
  - Python
  - Ruby
  - R
  - Javascript

Java Language

- Strongly typed programming language
  - All data (variables and constants) must **ALWAYS** be **declared** along **with** its **type**.
    - Identify the memory location by symbol name
    - Identify the memory contents by data type
- Declaration Examples:

  DataType SymbolName

  1. int age;
  2. String name;
  3. public class Person { }

Java Data Type Categories

- Only Two Data Type Categories

1. Pre-Defined Primitive data types:
2. Definable Reference data types:

Java Primitive Data Type

- Primitive data types:
  - Fundamental **predefined** data types
  - Passed by Value
    - Data value is **copied** and passed as a parameter therefore the **original data value cannot be changed when passed by value**

Java Reference Data Type

- Reference data types:
  - Classes and Objects are **definable aggregates**
  - Passed by Reference (like a pointer)
    - Reference is copied and passed as a parameter but always references the **same data object**

” The reference values (often just *references*) are pointers…”

https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.3.1

Java Primitive Data Types

1. **byte** *8-bit integer* (2^7 to 2^7 minus 1, i.e. -128 to 127)
2. **short** *16 bit integer* (2^15 to 2^15 minus 1, i.e. -32,768 to 32,767)
3. **int** *32 bit integer* (-2^31 to 2^31 minus 1)
4. **long** *64 bit integer* (-2^63 to 2^63 minus 1)
5. **float** *32-bit single precision floating point*
6. **double** *64-bit double precision floating point*
7. **boolean** *true or false*
8. **char** *16 bit Unicode character*

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Java Primitive Data Type Use

```
int n = 0;    // declare, create, init int value 0


n = 7;        // overwrite int value with 7;
n++;          // increment int value by 1
n = n + 1;    // increment int value by 1
```

Java Primitive Data Types

- “… **new** keyword isn't used when initializing a variable of a primitive type. Primitive types are special data types built into the language; they are not objects created from a class.“

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Java Primitive Data Types

- Literal values for primitive data types:
  1. byte b = 0;
  2. short s = 1000;
  3. int = 100000;
  4. long x = 0L;
  5. float y = 0.0f
  6. double z = 0.0d
  7. ‘\u0000’ for char
  8. false for boolean

Java Primitive Data Types

- Literal values for primitive data types:

```
int n1 = 13;              // 13 in decimal notation
int n2 = 0b1101;          // 13 in binary notation
int n3 = 0x0d;   // 13 in hexadecimal notation

double x1 = 123.4
double x2 = 1.234e2   // x1 in scientific notation
```

Java Primitive Data Types

- Literal values for primitive data types:

```
long creditCardNum = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
```

Java Primitive Data Types

- Literal values for primitive data types:

```
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes =
0b11010010_01101001_10010100_10010010;
```

Java Primitive Data Types

- PLACE "_" ONLY BETWEEN DIGITS

```
int x4 = 0_x52;          // INVALID
```

  - NEVER At the beginning or end of a number

```
int x2 = 52_;            // INVALID
int x5 = 0x_52;          // INVALID
```

  - NEVER Adjacent to a decimal point in a floating point literal

```
float pi1 = 3_.1415F; // INVALID
float pi2 = 3._1415F; // INVALID
```

  - NEVER Prior to an F or L suffix, example:

```
long socialSecurityNumber1 = 999_99_9999_L;
```

  - NEVER In positions where a string of digits is expected

Java Primitive Data Types

- Literal values for primitive data types:

```
char a1 = ‘A’;          // uppercase A character
char a2 = ‘a’;          // lowercase a character
char c1 = ‘\n’;         // newline character
char c2 = ‘\t’;         // tab character
```

Java Primitive Data Types

- Default values for primitive data types in class:
  1. 0 for byte
  2. 0 for short
  3. 0 for int
  4. 0L for long
  5. 0.0f for float
  6. 0.0d for double
  7. ‘\u0000’ for char
  8. false for boolean

Java Primitive Data Types

- Declaring variables of primitive data types without explicit initialization
  - Compiler set variables to reasonable default value

```
int age;                // initialized to 0
double gpa;             // initialized to 0.0d
char middleInitial;     // initialized to '\u0000'
```

Java Primitive Data Types

- Declaring and initializing variables of primitive data types

```
int age = 17;
double gpa = 4.0;
char middleInitial = ‘G’;
```

Java Reference Type

- A Class is a reference type
  - Definable custom data type
    - The fundamental Unit for Java Object Oriented Programming: Everything is a class
  - Wrapper for definable data and/or code
  - Aggregate data type
    - Including Primitive data types
    - Including Other reference types
    - Including Program code

Class Static members

- Use class statically
- Class members defined as ‘static’
  - ONE memory allocation
  - Program Scope
    - Always available for use
    - No need to create object with “new”

Class Object Instance members

- Create and use objects from class
- Class members defined without '**static**'
  - New memory allocation with each object created
  - Object Instance scope
    - DOES NOT EXIST UNTIL object is created with "new"
    - Java Garbage Collection (GC) automatically deletes objects when no longer needed.

Simple Class Name

```
public class Name {
        // state is one String
        public String n = ”Dan”;
}
```

- Class **Name** is a container class for a String
  - See class **Java.Lang.String**
- Class **Name** is a Reference Type
- Object instance Member data is a String named ‘**n**’ holding a String value

Use Simple Class Name

```
// create object on heap and assign reference to obj
Name obj = new Name(); // implicit default constructor
// use object on heap through reference in obj
System.out.println(obj.n);        // show #1 init state
obj.n = “Daniel”;                 // overwrite state
System.out.println(obj.n);        // #1 current state
Name obj2 = new Name();           // create object #2
System.out.println(obj2.n);       // show #2 init state
System.out.println(obj.n);        // #1 current state
```

Use Simple Class Name

CONSOLE OUTPUT

Dan

Daniel

Dan

Daniel

Simple Class Label

```
public class Label {
        // state is one String
        public static String n = ”Dan”;
}
```

- Class **Label** is a container class for a String
  - See class **Java.Lang.String**
- Class **Label** is a Reference Type
- Static class Member data is a String named ‘**n**’ holding a String value

Use Simple Class Label

```
// use class Label
System.out.println(Label.n);      // show init state
Label.n = “Daniel”;               // overwrite state
System.out.println(Label.n);      // show current state
Label.n = “Danny”;                // overwrite state
System.out.println(Label.n);      // show current state
System.out.println(Label.n);      // show current state
```

Use Simple Class Name

CONSOLE OUTPUT

Dan

Daniel

Danny

Danny

Java Reference Type

- A Class is a reference type
  - To instantiate an object from a class:
  1. Using keyword "**new**"
  2. Calling a class constructor

- Must Create ALL Objects with "**new**"
  - **EXCEPT** String objects

Java Reference Type

- To Create a Person object:

Person dan = null;

dan = new Person();

- Data Type is "**Person**" class
- Variable Name (Identifier) is "**dan**"
- Class constructor is "**Person()**"

Java Reference Type

- To Create a Student object:

Student sam = new Student();

- Data Type is "**Student**" class
- Variable Name (Identifier) is "**sam**"
- Class constructor is "**Student()**"

Java Reference Type

- To Create a container object:

List<String> names = null;

names = new ArrayList<>();

- Data Type is "**List<String>**" interface
- Variable Name (Identifier) is "**names**"
- Class constructor is "**ArrayList<>()**", where <String> is compiler inferred

Java String: Java Reference Type

- Character String

  “This is a LITERAL character string.”

- A String is a Reference Type

  java.lang.String class

- A String is immutable
- **NOT** an array of characters terminated by a null character (C Language).
  - A Java String object is **NOT** a C language string.

Java String

- Special String treatment:
  - Enclosing characters in double quotes **automatically** creates a String object:

  String **name** = “**Dan**”;

  - Identifier ”**name**” contains a reference to a String object containing the immutable value of “**Dan**”.

Java String

- For String objects, Use of the '*new*' keyword is optional (and **discouraged**)
  - Reference:
    - Java ***string pool*** and ***string interning***.
  - Both memory (and it's allocation time) are conserved by saving immutable strings in a pool. When a new string is created, **if it is a repeated string**, a reference to an already preserved immutable string in the pool is established in lieu of a new created string.

Java String

- Use of the ‘new’ keyword is optional (and discouraged) for creating String objects.
- DO

  String s = “abc”; // allows interning
- DO NOT

  String s = new String(“abc”); // forces new string

- String objects

Array: Java Reference Type

- To Create a fixed size array container object:

int [] myArray = new int[3];

- Data Type is "**int []**" int array
- Variable Name (Identifier) is "**myArray**"
- The array is created for ONLY three integers by using "**int[3]**"

Array: Java Reference Type

"In the Java programming language, *arrays* are objects…"

https://docs.oracle.com/javase/specs/jls/se8/html/jls-10.html

"An *object* is a *class instance* or an *array*. "

https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.3.1

Java Reference Type

- To Create a fixed size array container object:

int [] myArray = { 1, 2, 3 };

- Data Type is "**int []**" int array
- Variable Name (Identifier) is "**myArray**"
- The array is created for ONLY three integers by using the initializer "**{1,2,3}**"

Java Reference Type

- To Create a fixed size array container object:

String [] myArray = { “1”, “2”, “3” };

- Data Type is “**String []**” String array
- Variable Name (Identifier) is “**myArray**”
- The array is created for ONLY three Strings by using the initializer

Java Pass Primitives By Value

- Primitive data types are int, double, etc.
- Memory for Primitive data types are allocated on the stack
- Copies of Primitive data types are passed to methods
- Methods CAN NOT modify the Original primitive data type.

Java Pass Object Reference By Reference

- Objects are Reference Types
- References point to Object allocation in heap memory
  - TWO memory allocations are needed to use an object.
    1. Object allocated on the heap
    2. Reference (pointer) allocated on stack, pointing to Object allocation on the heap
- References passed to methods are copies
- Copies STILL POINT TO SAME OBJECT

Simple Class N

```
public class N {
        public int n = 0; // state is one int
}
```

- Class N is a container class for an integer
  - See class **Java.Lang.Integer**
- Class **N** is a Reference Type
- Object instance Member data is an integer named '**n**' holding an integer value

sillySwap method

```
public void sillySwap(N o1, N o2) {
        N temp = o1;      // save for later


        System.out.println("Swap object references:");
        o1 = o2;
        o2 = temp;          // original o1
        // COPIES of references have changed
}
```

showObjects method

```
// output the state of each object on console
public static void showObjects(N o1, N o2)
{
   System.out.println(”  " + o1.n + " " + o2.n);
}
```

Use SillySwap method

```
public void sillySwapObjects() {
   N o1 = new N();      // create object 1
   N o2 = new N();      // create object 2
   o1.n = 1;            // set value 1 in object 1
   o2.n = 2;            // set value 2 in object 2
   ValueN.showObjects(o1, o2);    // 1 2
   ValueN.sillySwap(o1, o2);  // useless swap
   ValueN.showObjects(o1, o2);    // 1 2
}
```

Use SillySwap method

Swap object references produces:

Console Output:

1 2

1 2

smartSwap method

```
public void smartSwap(N o1, N o2) {
        N temp = new N();
        temp.n = o1.n     // save for later


        System.out.println("Swap object state:");
        o1.n = o2.n;
        o2.n = temp.n;    // original o1 state
        // state of Objects have changed
}
```

Use smartSwap method

```
public void smartSwapObjects() {
   N o1 = new N();      // create object 1
   N o2 = new N();      // create object 2
   o1.n = 1;            // set value 1 in object 1
   o2.n = 2;            // set value 2 in object 2
   ValueN.showObjects(o1, o2);    // 1 2
   ValueN.smartSwap(o1, o2);  // swap state
   ValueN.showObjects(o1, o2);    // 2 1
}
```

Use SmartSwap method

Swap object state produces:

Console Output:

1 2

2 1