

2

## Objects and Values

Cmput 114 - Lecture 4  
Department of Computing Science  
University of Alberta  
©Duane Szafron 1999

## About This Lecture

- In this lecture, we will learn about values and how they differ from objects.
- We also learn how to do simple computations with values.

©Duane Szafron 1999

3

## Outline

- Objects versus values
- Pure and hybrid languages
- Java values and primitive types
- Some Java operators

©Duane Szafron 1999

4

## Objects versus Values

- Some real world entities are so simple that we represent them by values instead of by objects.
- A value has no protocol, so it cannot respond to messages.
- However, values can be used as arguments in messages.
- Values can also be returned as the results of messages.

©Duane Szafron 1999

5

## Pure Paradigm Languages

- How simple is simple enough to represent as a value?
- The answer depends on the programming language used.
- In "pure" procedural programming languages like C and Pascal, there are no objects, only values.
- In "pure" object-oriented languages like Smalltalk, there are no values, just objects.

©Duane Szafron 1999

6

## Hybrid Programming Languages

- Languages with both objects and values are object-procedural hybrid languages.
- Hybrid languages differ on what is an object and what is a value.
- In Java, numbers can be represented by values or objects while in C++ they are values.
- In Java, strings are objects while in C++ they are values.
- In both languages, Streams are objects.

©Duane Szafron 1999

7

## Simple Program in Java and C++

- If our simple program is expressed in **Java**, then the “Hello” message argument is an **object**.
- If our simple program is expressed in **C++**, then the “Hello” message argument is a **value**.

The diagram shows a central box labeled "screen" with a "print" message. Two arrows point to it: one from a box labeled "Java" and another from a box labeled "C++". The arrow from "Java" points to an oval containing the string "Hello". The arrow from "C++" points to a box containing the string "Hello". A callout box points to the "screen" box with the text: "This is a PrintStream object in both Java and C++".

©Duane Szafron 1999

8

## Java Values - Primitive Types

- We group similar kinds of Java values together so that every value has a **primitive type**.
- For example, the numbers 3, 4 and 5 have primitive type **int**.
- Since a value is not an object, it does not have a **class**.
- To avoid saying "the class of an object or the primitive type of a value", we use the word **type** to mean class or primitive type.

©Duane Szafron 1999

9

## Java Values

- Here are some of the primitive types for Java values:

int	43 -12 9999999
char	'H' '\n' (newline) '\t' (tab)
float	43.0f -12.5f 9.5E-5f
boolean	true false

©Duane Szafron 1999

10

## Computing with Values

- Since we cannot send messages to values, how can we do any computing with values?
- There are three ways to use values:
  - use them as arguments to messages
  - return them as the results of a message
  - use operators on them

©Duane Szafron 1999

11

## Using Values as Arguments

The diagram shows a sequence of messages and values. A box labeled "print" with the value "50" has an arrow pointing to an oval containing "50". This oval has an arrow pointing to a box labeled "println" with the value "60". The "println" box has an arrow pointing to another oval containing "50". This second oval has an arrow pointing to a box labeled "print" with the value "70". The "print" box has an arrow pointing to a final oval containing "5060" and "70".

©Duane Szafron 1999

12

## Using Values as Results

The diagram shows a box labeled "length" with an arrow pointing to an oval containing the string "Hello". From the oval, an arrow points to a box containing the number "5".

©Duane Szafron 1999

13

## Java Operators

- Values can be manipulated using operators.
- Operators are not messages!
- For example, there are some arithmetic operators that take numerical operands and compute numerical results.

©Duane Szafron 1999

## Object Creation, Object References and Variables

Cmput 114 - Lecture 5  
Department of Computing Science  
University of Alberta  
©Duane Szafron 1999

15

## About This Lecture

- In this lecture we will learn how to create objects in our language independent diagram world.
- We will learn about various kinds of object references that can be used to identify the objects that we want to use.
- We will also study different kinds of variables including static variables and local variables.

©Duane Szafron 1999

16

## Outline

- Object creation and Date example
- References
  - Literals
  - Values
  - Variables
  - Constants
- Static variables
- Local variables
- Message Parameters

©Duane Szafron 1999

17

## Object Creation is Needed

- When we express a computation using a diagram, we never have to create any objects, we just draw them and then send messages to them.
- In a written program, we must provide some instructions to create objects before we can send any messages to them.

©Duane Szafron 1999

18

## Object Creation

- Every object must be created before it can be used.
- An object creation primitive creates a new "empty" object.
- A custom message called a constructor is sent to the object to initialize its state.
- A class may have more than one constructor.
- In many languages, the name of the constructor is the name of the class.

©Duane Szafron 1999

19

### Partial Constructor Protocol - Date

- All constructors have the same name, but differ in the types of the argument objects.

- Set the new Date to today.
- Set the new Date to the Date represented by the argument String.

©Duane Szafron 1999

20

### Object Creation - Date

Two different constructors.

©Duane Szafron 1999

21

### Object References are Needed

- In a diagram, we can send a message to any object that we have drawn simply by pointing a message arrow at it.
- In a written program, we need to have some notation for referring to objects so we can send messages to them.
- If we create an object and don't have a reference to it, we can never use it.

©Duane Szafron 1999

22

### Object References - Literal

- An object reference is a language expression that refers to an object.
- The simplest kind of object reference is a literal object reference.
- A literal object reference refers to the same object at all times.
- You can think of a literal object reference as a nameplate attached to an object.

©Duane Szafron 1999

23

### Literal Object References - String

literal object reference

Literals cannot be re-bound

©Duane Szafron 1999

24

### Value References

- In a hybrid language, a value reference is a language expression that refers to a value.
- The simplest kind of value reference is a literal value reference that refers to the same value at all times.
- You can think of a literal value reference as a nameplate attached to a value.
- A literal is either a literal object reference or a literal value reference.

©Duane Szafron 1999

25

## Literal Value References - ints

literal value reference

Literals cannot be re-bound

©Duane Szafron 1999

26

## Variables

- A variable object reference is an object reference that may refer to different objects at different times.
- A variable value reference is a value reference that may refer to different values at different times.
- A variable is either a variable object reference or a variable value reference.
- A variable is said to be bound to its object or value.

©Duane Szafron 1999

27

## Re-binding Variables

- A variable can be re-bound to a different object or value.
- You can think of a variable as a nameplate that can be moved.
- However, in some languages, a variable is restricted to objects or values of a particular type called its declared type.
- More than one variable can be bound to the same object or value at the same time.

©Duane Szafron 1999

28

## Variables - Example

More than one variable bound to the same object.

variable object reference

variable value reference

Rebinding a variable

©Duane Szafron 1999

29

## Constants

- A constant is like a variable, but once bound to an object or value it cannot be rebound.
- It is different than a literal since:
  - it is not restricted to those types that have literals
  - its language notation is like a variable

©Duane Szafron 1999

30

## Constants - Example

constant object reference

constant value reference

Constants cannot be re-bound

©Duane Szafron 1999

## Kinds of Variables

- There are four kinds of variables:
  - static (class) variables
  - local (temporary or method) variables
  - message parameters
  - instance (object) variables
- Every variable has two characteristics.
- The scope or visibility is the region of program code that can use the variable.
- The lifetime is the time that the variable exists (can be used).

©Duane Szafron 1999

## Static Variables

- A static variable or class variable is declared in a class.
- Its lifetime is the entire time that the program is run.
- Its scope is either:
  - public (the entire program)
  - private (the class where it is declared)
- Static constants are also allowed.

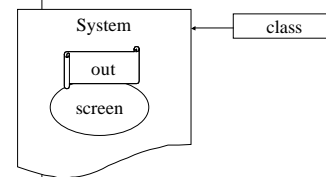
©Duane Szafron 1999

## Public Static Variables

- There may be three different types involved with a public static variable.
- Recall that the declared type of a variable is the type of object or value to which it can be bound.
- The exporting class is the class in which it is declared.
- The using class is the class in which it is used to reference an object or value.

©Duane Szafron 1999

## Public Static Variables - out



The exporting class of out is System.  
The declared type of out is PrintStream.  
It is actually a constant.

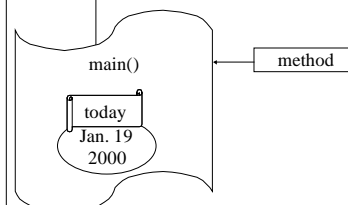
©Duane Szafron 1999

## Local Variables

- A local variable or temporary variable or method variable is declared in a block of code called a method.
- Its lifetime is the time that the method is running.
- Its scope is the method it is declared in.
- Local constants are also allowed.

©Duane Szafron 1999

## Local Variables - Example



©Duane Szafron 1999

37

## Message Parameters

- A message parameter is declared at the start of a block of code called a method.
- A message parameter is bound to an argument object or value when the method is called and cannot be re-bound.
- Its lifetime is the time that the method is running.
- Its scope is the method it is declared in.

©Duane Szafron 1999

38

## Message Parameters - Example

The diagram shows a message box for the method `concat(String aString)`. Inside the message box, there is a parameter box labeled `aString` which contains the value `"Flintstone"`.

©Duane Szafron 1999

39

## Instance Variables

- We will discuss instance variables in a future lecture.

©Duane Szafron 1999

40

## General Object References

- Recall that an object reference is any language expression that refers to an object and a value reference is any language expression that refers to a value.
- There are many other kinds of references in addition to literals, variables and constants.
- For example, since a message expression returns an object or value, the message expression itself is also a reference.

©Duane Szafron 1999

41

## Message Expression Object Reference

The diagram shows a sequence of operations: a message `toUpperCase` (labeled 1) is sent to the object `"Hello "`, resulting in the object `"HELLO "`. This result is then passed to a `trim` message (labeled 2), which produces the final object `"HELLO"`. A callout box with a curved arrow pointing to the `toUpperCase` message states: "This message expression is an object reference, so another message can be sent to the result".

©Duane Szafron 1999