Welcome to CIS113

Data Structures

Introductions and Chapter 1

Watch eclipse-workbench lesson 1

Four Types of Flow of Control (Review)

- **Sequential Processing**
  - **Execute instructions in order**
- Method Call
  - Jump to code in method, then return
- Selection
  - Choose code to execute based on data value
- Looping or Iteration
  - Repeat operations for multiple data values

Sequential Processing

- The pseudocode for calculating the sum of two numbers would look like this:

```
read first number
read second number
set total to (first number +
                second number)
output total
```

Code it

- Create a new project: Addition
- Name the class: Addition
- Instantiate a variable "scan" of type "Scanner", passing "System.in" as the argument

  ```
  Scanner scan = new Scanner(System.in);
  ```

  Note: try to find a solution to the error through the eclipse IDE
- Prompt the user for a number using the print method of the System output class

  ```
  System.out.print("Enter integer #1: ");
  ```
- Read the users input using the nextInt method of object scan

  ```
  int num1 = scan.nextInt();
  ```
- Prompt the user for a second number using the print method of the System output class

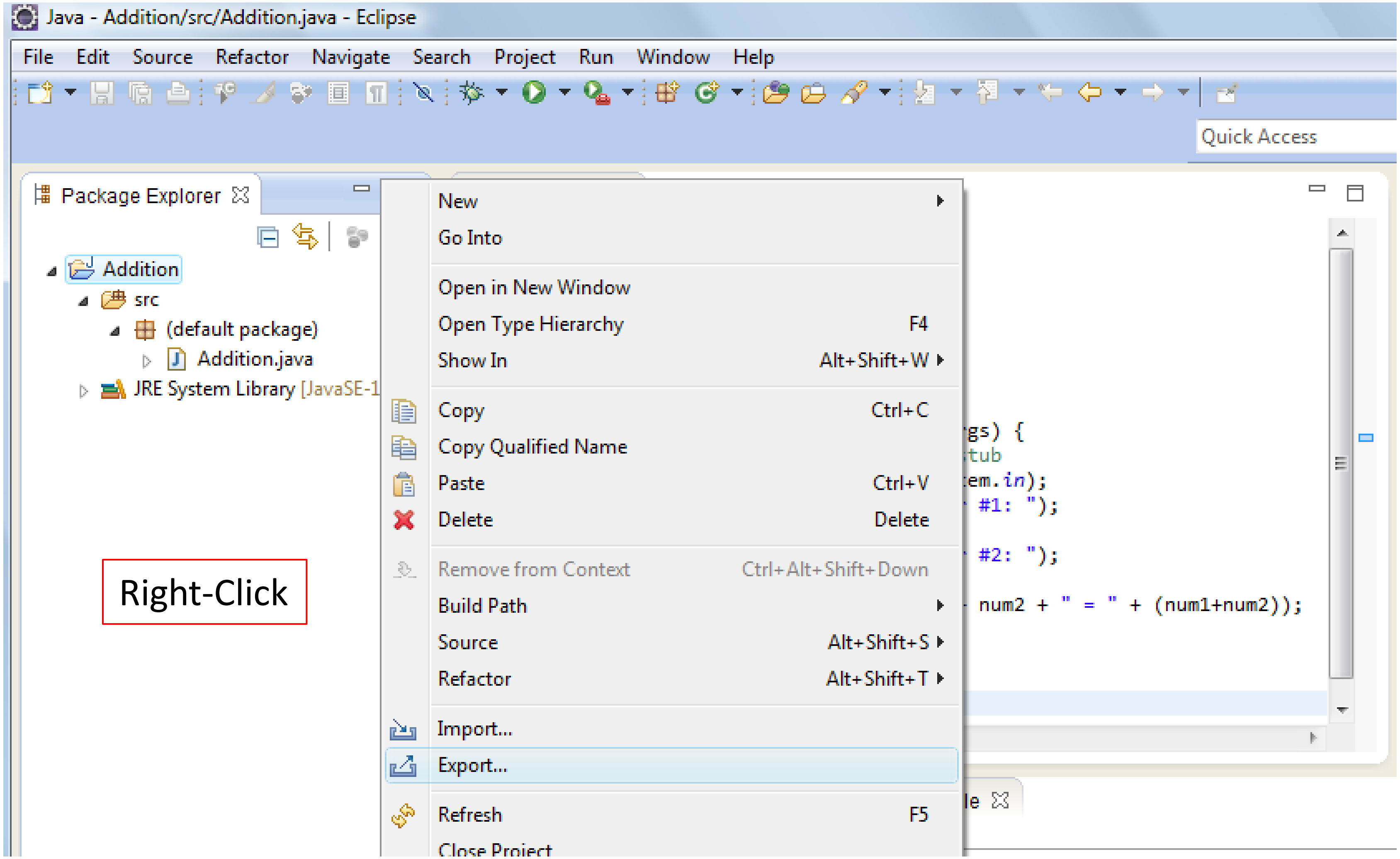
  ```
  System.out.print("Enter integer #2: ");
  ```
- Read the users input using the nextInt method of object scan

  ```
  int num2 = scan.nextInt();
  ```
- Output the result

  ```
   System.out.print(num1 + " + " + num2 + " = " + (num1+num2));
  ```
- Free resources

  ```
  scan.close();
  ```

Export It

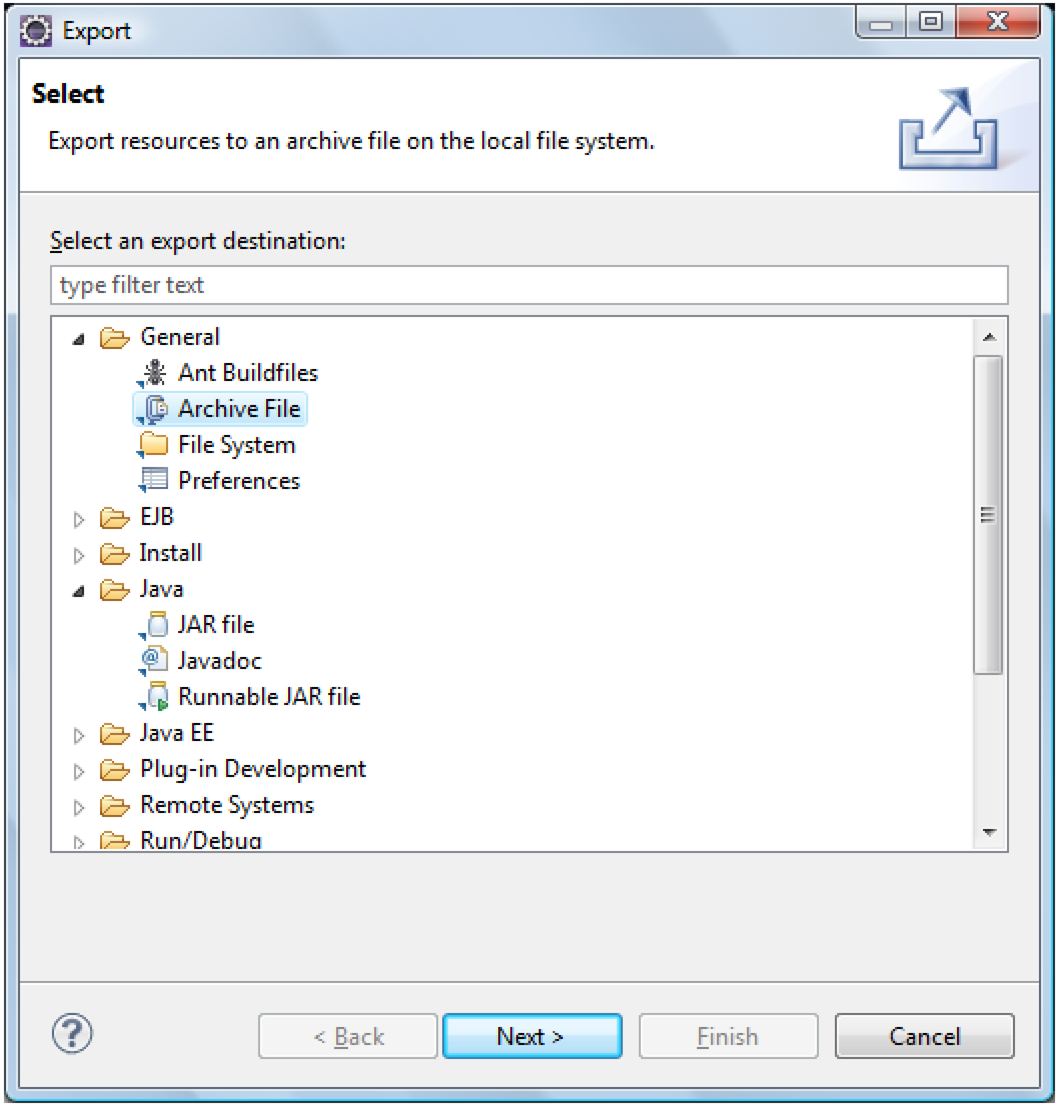


```
gs) {
tub
em.in);
 #1: ");

 #2: ");

 num2 + " = " + (num1+num2));
```

Right-Click



Export

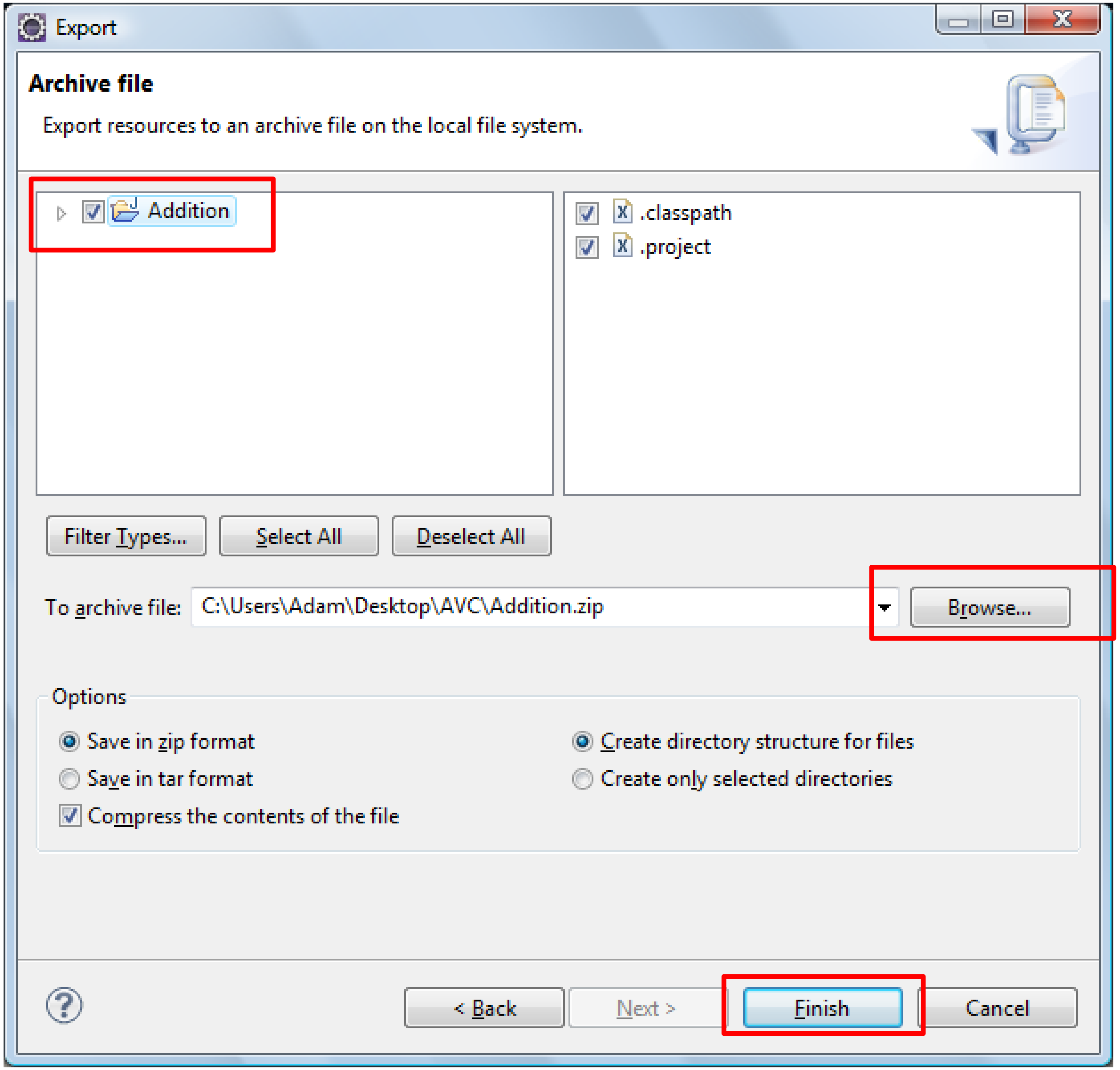
Select

Export resources to an archive file on the local file system.

Select an export destination:

type filter text

- General
  - Ant Buildfiles
  - Archive File
  - File System
  - Preferences
- EJB
- Install
- Java
  - JAR file
  - Javadoc
  - Runnable JAR file
- Java EE
- Plug-in Development
- Remote Systems
- Run/Debug



Export

Archive file

Export resources to an archive file on the local file system.

☑ Addition

☑ .classpath

☑ .project

To archive file: C:\Users\Adam\Desktop\AVC\Addition.zip

Options

● Save in zip format

○ Save in tar format

☑ Compress the contents of the file

● Create directory structure for files

○ Create only selected directories

Congratulations, now you have zipped the project file using eclipse!

What makes good software?

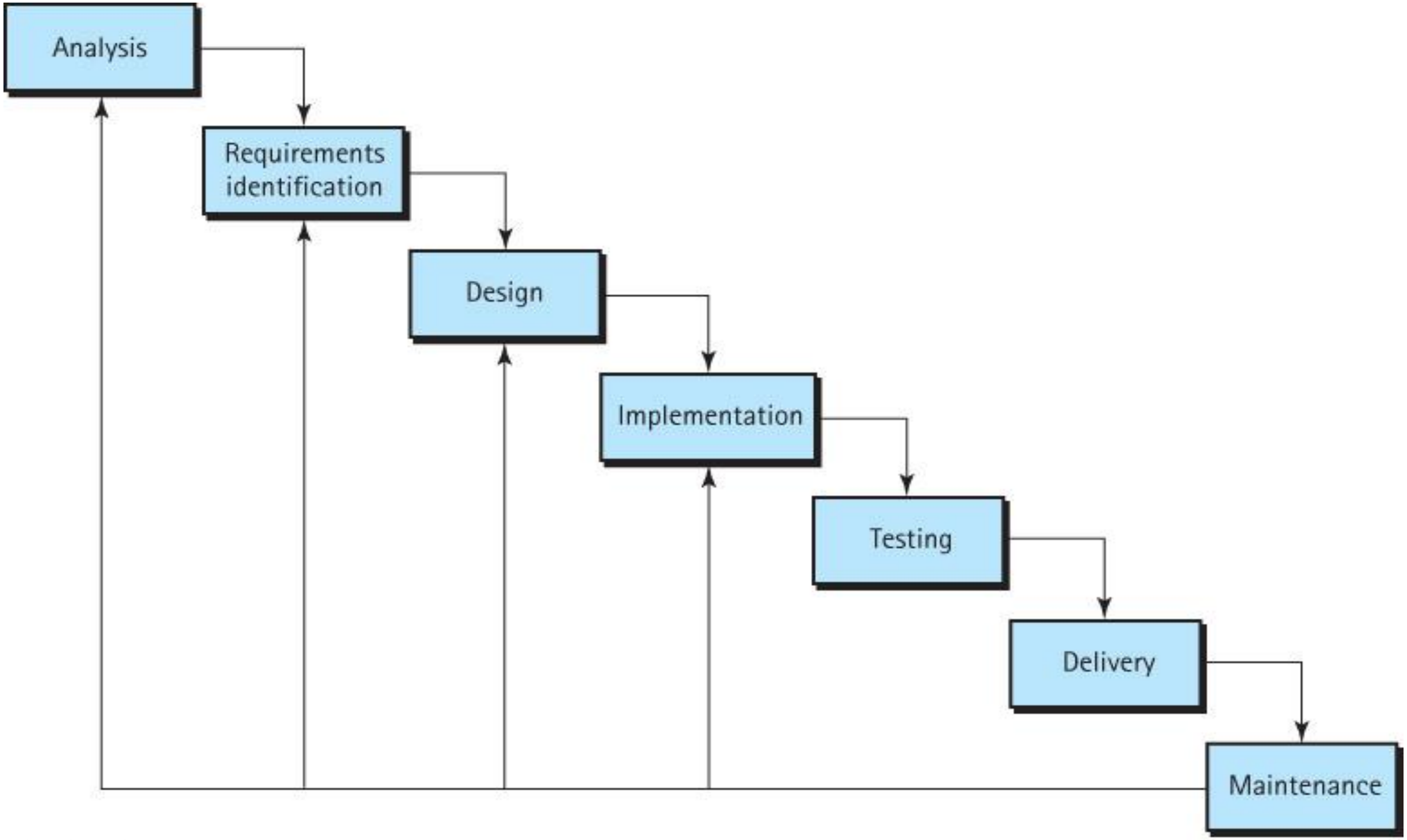
(think perspectives: user/developer)

- It works!
  - Complete & correct
- It is user friendly.
- It is efficient.
  - Completes the task in a reasonable amount of time
- It is easily modified.
  - Little time
  - Low effort
- It is reusable.
  - think object oriented classes and methods
- It can be completed on time and within budget…?

Software Engineering

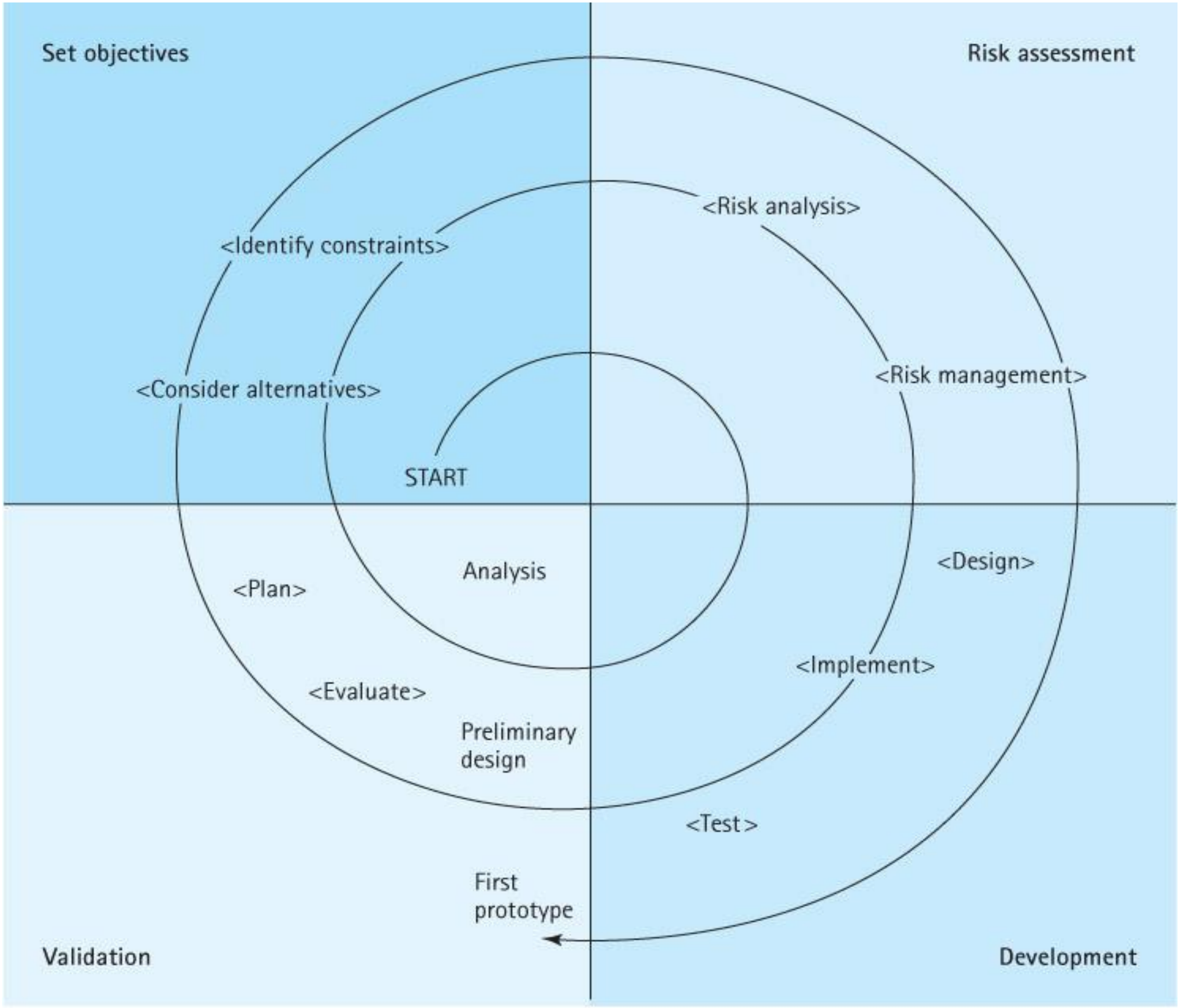
- Cowboy programming
  - Lacks methodology
- The Waterfall Method (pages 3-4 in the text)
  - Rigid
  - Heavy on documentation
  - Is one method for all problems is unrealistic?
  - Big Bang delivery
- Agile Methods
  - Flexibility is introduced
  - Customer involvement
  - Incremental delivery
  - Flexible and evolving
  - Developers working in pairs -> fewer defects
    - Is it worth the $$$
- The Unified Method

Waterfall Life-Cycle Model



(a) Waterfall model

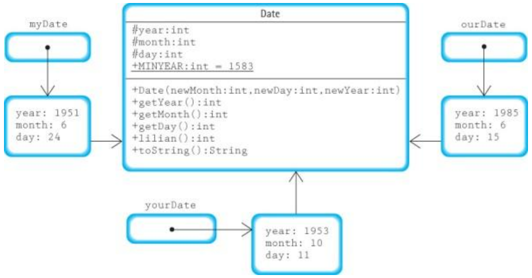
Spiral Life-Cycle Model



(b) Spiral model

The Unified Method

- Use cases
  - Actions required to complete a task
- Architecture-centric
  - Considers the overall structure of the system and how the systems' components interact
- Iterative & incremental
  - Builds upon itself, using the early stages as building blocks
- UML – Unified Modeling Language
  - Visualize the program



Object Oriented (OO) Programming

- Primitive data types
  - byte, char, short, int, long, float, double, and boolean
- Class – defines structures of its' object(s)
- Object – run-time entities used by applications
  - Constructor
  - Observer
  - Transformer
- Information hiding: "only show what is necessary"
  - public
  - protected
  - package
  - private

OO – Terminology

- Constructor
  - An operation that creates a new instance of a class
- Observer
  - An operation that allows us to observe the state of an object without changing it
- Transformer
  - An operation that changes the internal state of an object

OO – Inheritance

- Types of classes
  - superclass
  - subclass
    - Inherits properties (data and action) from the superclass
- Variable Access modifiers
  - public
    - Visible everywhere
  - protected
    - Visible to subclasses in the same package and other packages
  - package (default!)
    - Visible to subclasses within the same package
  - private
    - Visible within the class

Good practices: only transformers should change data within a class!

Objects

- Created from classes at runtime
- Can contain and manipulate data
- Many objects can be created from the same class
- Instantiating objects uses the ‘new’ operator

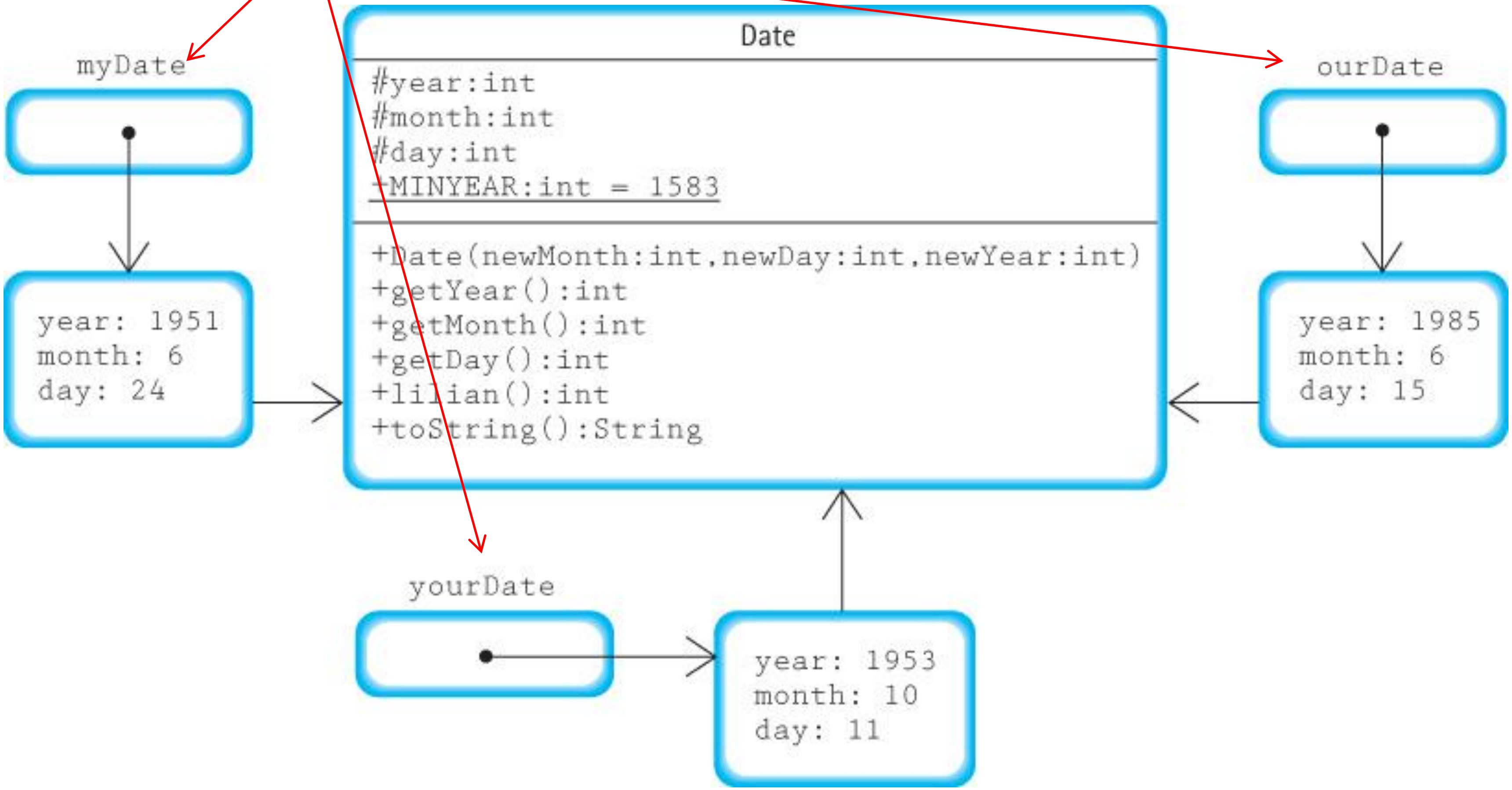
```
Date myDate = new Date(6, 24, 1951);
Date yourDate = new Date(10, 11, 1953);
Date ourDate = new Date(2, 6, 2012);
```

Vars myDate, yourDate, and ourDate reference objects of the class date.

UML: Objects

```
Date myDate = new Date(6, 24, 1951);
Date yourDate = new Date(10, 11, 1953);
Date ourDate = new Date(6, 15, 1985);
```

These variables reference the Date object



If no object has been instantiated for a variable, its memory location holds the value “null”

Instantiating vs. Invoking an object

Instantiate

```
   Date ourDate = new Date(10, 11, 1953);
```

Invoke:

```
   int Year = ourDate.getYear();
```

Invoking the string function, automatically changes an object to a string

```
Date ourDate = new Date(2, 6, 2012);
System.out.println(“Today is “ + ourDate);
Today is 2/6/2012

System.out.println(“The year is “ + Year);
The year is 2012
```

Inheritance

- Allows user to create a new class (subclass) which is a specialization of another class (superclass)

extends indicates inheritance

```
public class IncDate extends Date
{
    public IncDate (int newMonth, int NewDay, int newYear)
    {
        super(newMonth, newDay, newYear);
    }
```

Java does not inherit constructors
(Requires new constructor)

super: reserve word indicating superclass

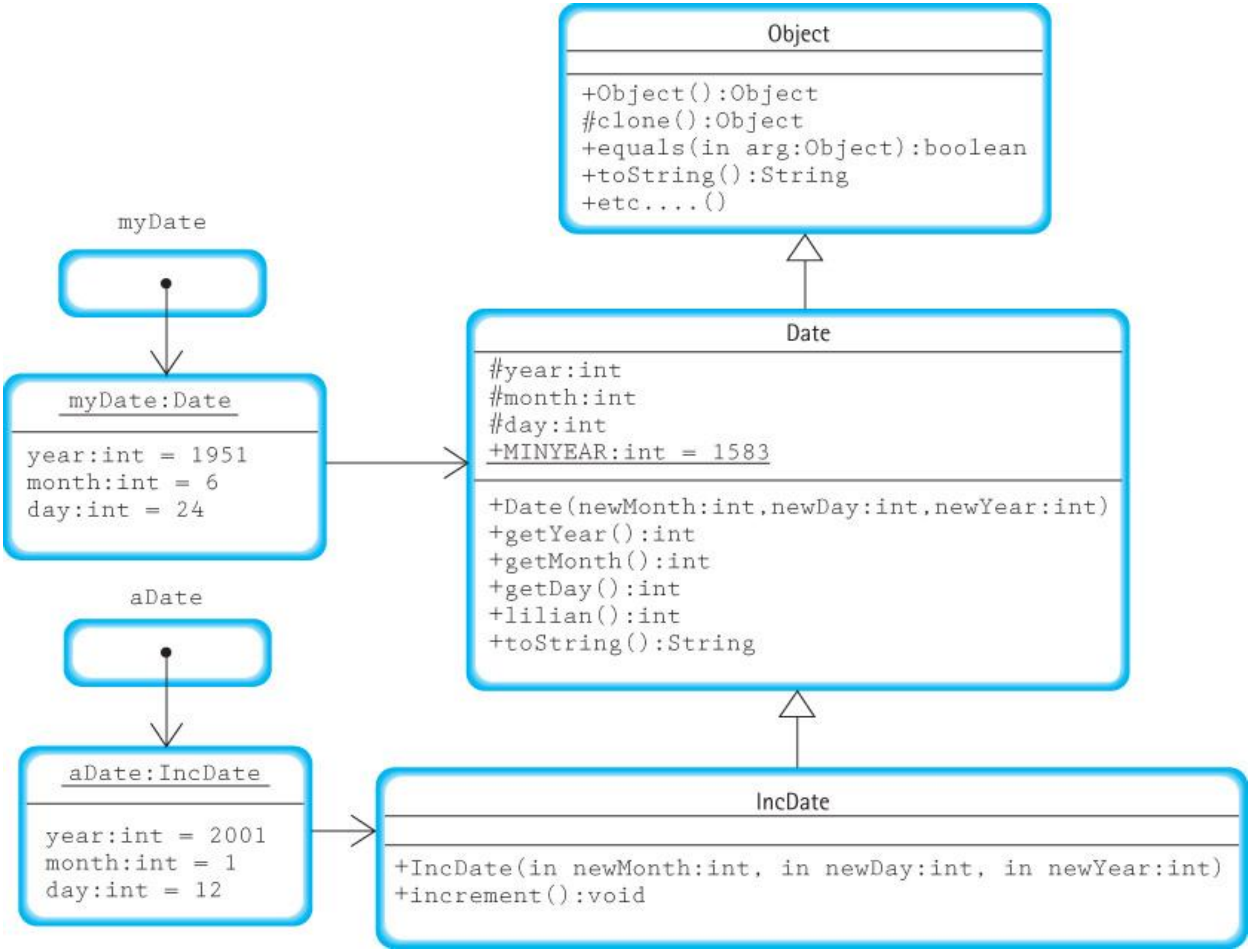
```
    public void increment()
    { // increment algorithm goes here }
}
```

Is IncDate an observer, constructor, or transformer?

The method is invoked in the code by…

```
IncDate testDate;
// assign some date to testDate
testDate.Increment();
```

UML: Inheritance



Object

+Object():Object
#clone():Object
+equals(in arg:Object):boolean
+toString():String
+etc....()

myDate

myDate:Date

year:int = 1951
month:int = 6
day:int = 24

Date

#year:int
#month:int
#day:int
+MINYEAR:int = 1583

+Date(newMonth:int,newDay:int,newYear:int)
+getYear():int
+getMonth():int
+getDay():int
+lilian():int
+toString():String

aDate

aDate:IncDate

year:int = 2001
month:int = 1
day:int = 12

IncDate

+IncDate(in newMonth:int, in newDay:int, in newYear:int)
+increment():void

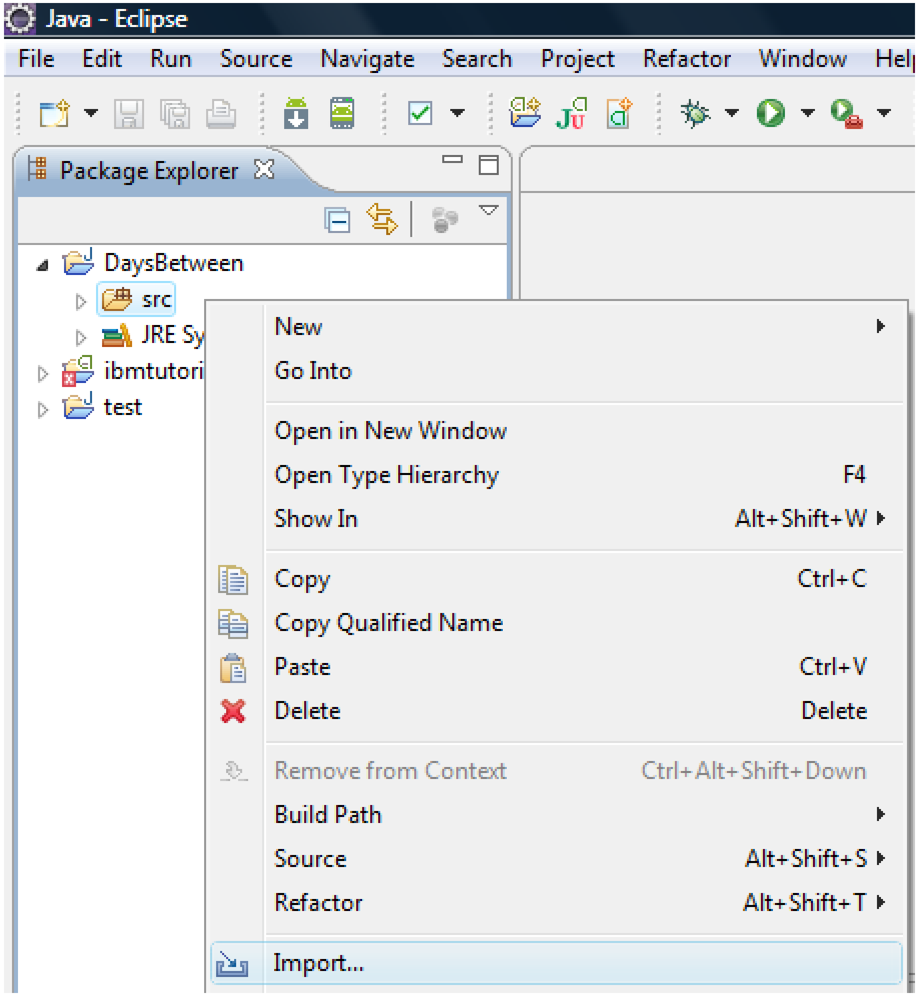
Break

- Watch eclipse totalbeginnerlesson01

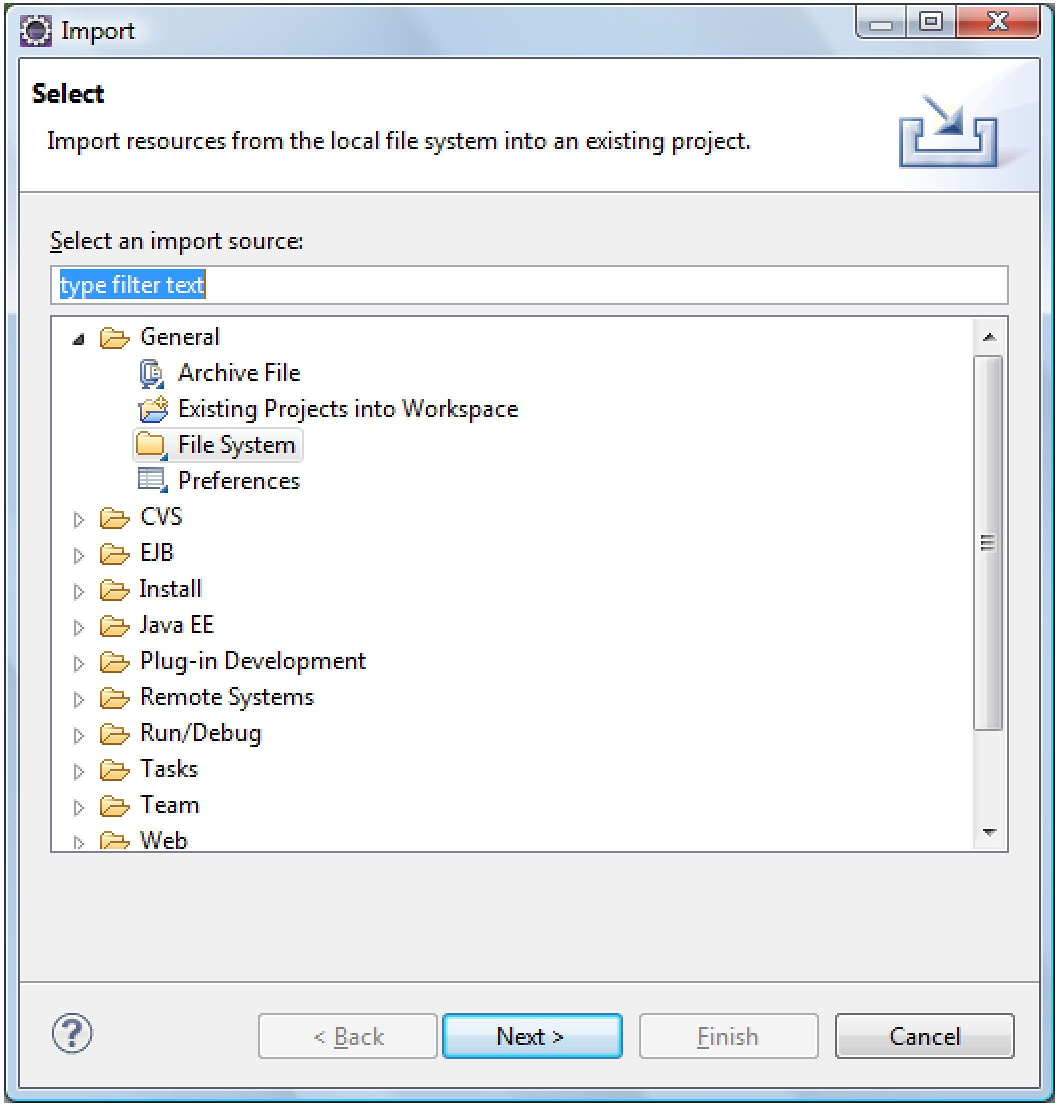
Lab: book problems 17 & 20

1. First Create a new Java Project in eclipse
2. Name it DaysBetween

Right-Click on the src subfolder of DaysBetween



Import – General – File System



Import

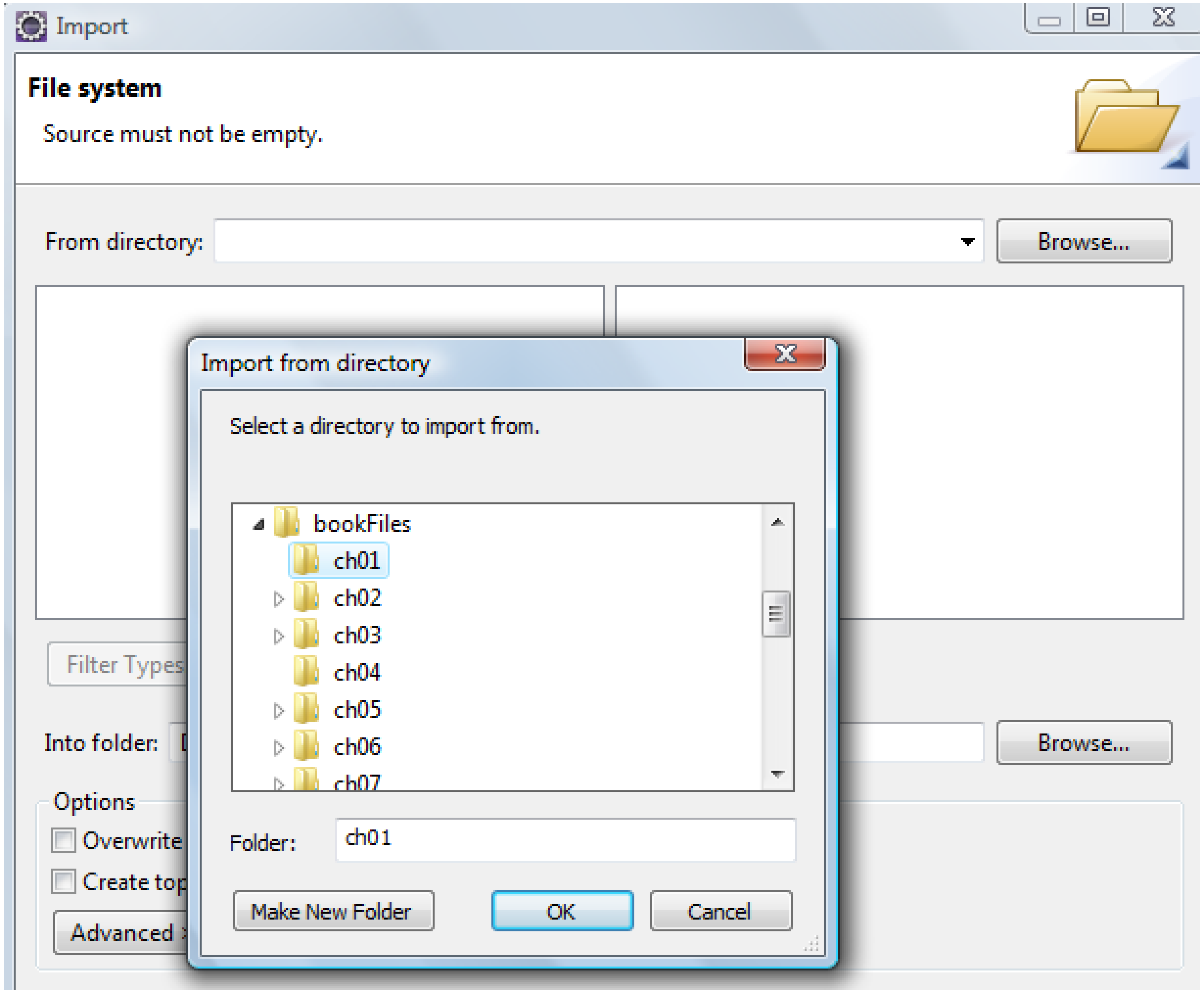
Select

Import resources from the local file system into an existing project.

Select an import source:

type filter text

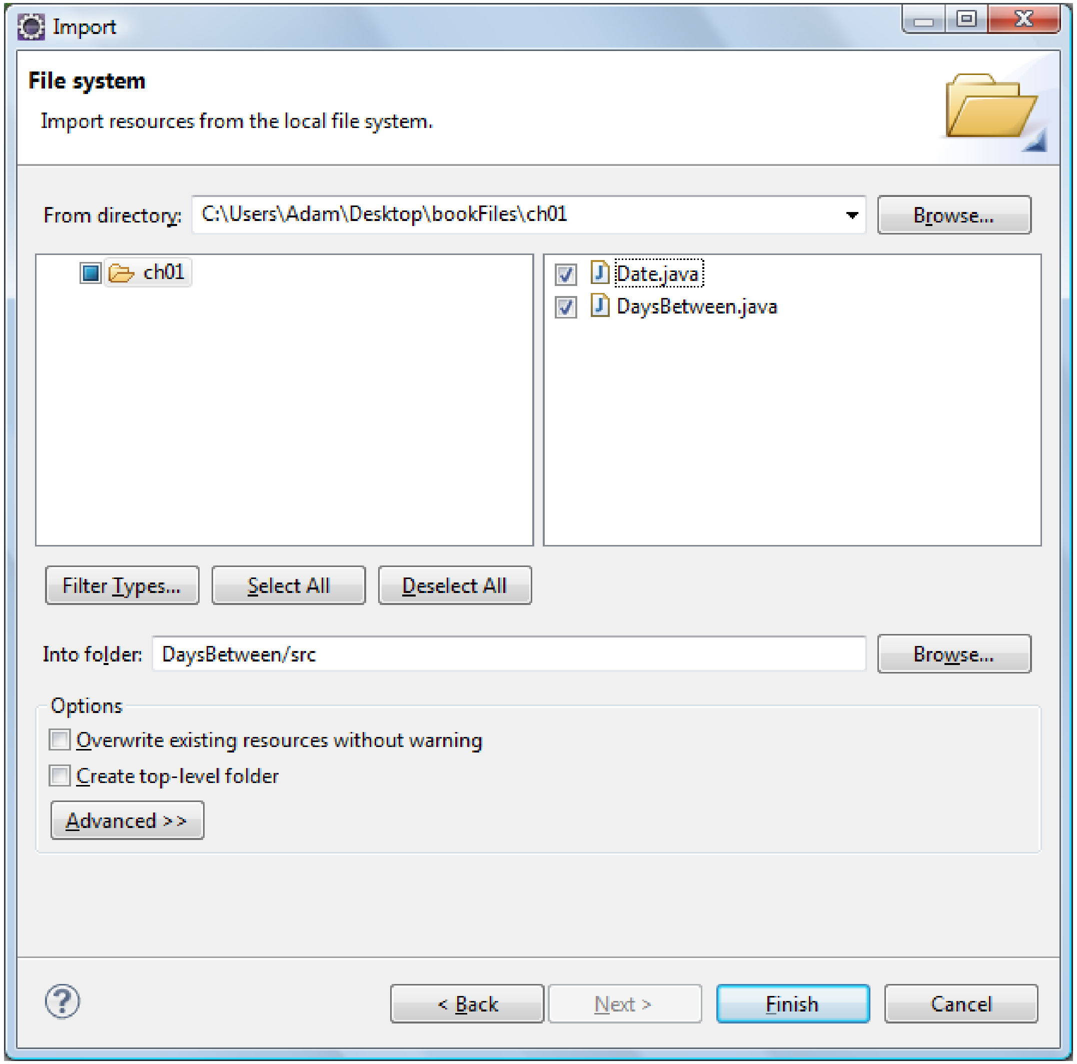
Browse to bookFiles ch01



Folder: ch01

Note: CSE113_SourceCode.zip on myAVC contains bookFiles

Select both files and click finish



Import

File system

Import resources from the local file system.

From directory: C:\Users\Adam\Desktop\bookFiles\ch01

■ ch01

☑ Date.java

☑ DaysBetween.java

Into folder: DaysBetween/src

Options

□ Overwrite existing resources without warning

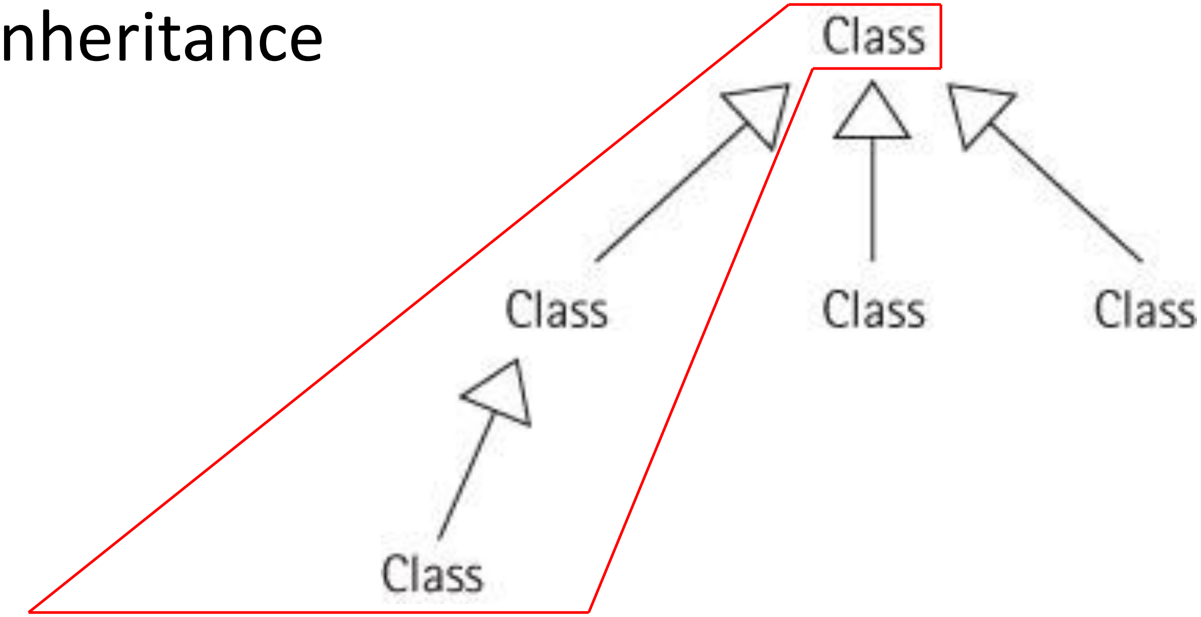
□ Create top-level folder

Lab

- Implement book problems 17 and 20 in the DaysBetween project

Inheritance Tree

- C++ (generally accepted as industry standard)
  - Multiple Inheritance



- Java
  - Supports single inheritance only

Packages

- File organization
- May be compiled separately
- Can be imported into different programs
- Easier to implement common class files
- Help to avoid naming conflicts
  - Two classes can have the same name if they are in different package

Syntax:

```
package packageName;                    // appears as the first line in the file
// declarations appear after specifying the package name
// all non-public classes within a package are hidden from the world outside the package
```

Import:

```
import packageName.*;                   //  imports all classes in packageName
import packageName.className            // imports only the class className in packageName

// Providing a directory is defined in your systems ClassPath
// This example expects the ClassPath system variable to contain a valid path to ch03.
import ch03.stacks.*;
```

File and method naming

- File name must match the public class
  - In terms of packages
    - Only one public class per package

```
package demo;
public class One{…}
class Two{…}
```

(The file must be named One.java)

Organization

- Bottom Line…
  - Regardless of…
    - File management
    - Package management
    - Coding indents (tabs)
    - Use of other white space (spaces and line breaks),
    - Instructions
    - and Comments

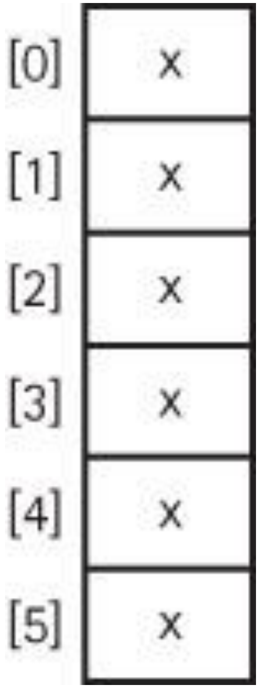
BE CONSISTENT

Data Structures

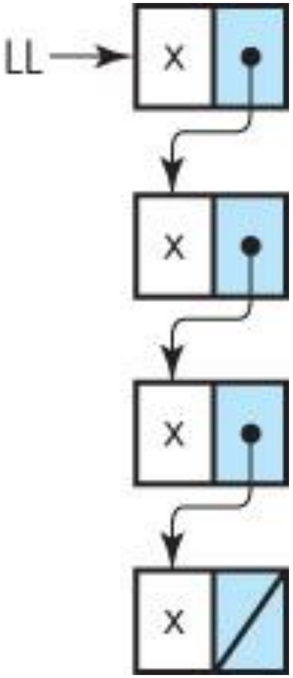
- Two of the key concepts in Computer Science
  - Organization
  - Efficiency
- Common indexing systems
  - Alphabetical / numerical ordering
  - Dewey Decimal System
- Indexing systems of Computer Science
  - Implementation dependent
    - Building blocks for other data structures
      - Array
      - Linked List
  - Implementation independent (more abstract)
    - Stack (LIFO – Last In First Out)
    - Queue (FIFO – First In First Out)
    - Sorted List (Telephone book and the dictionary – yes an array!)
    - Tree (Parent-child relationship, parent has no root)
    - Graph (Nodes, vertices, and edges)

Implementation Dependent Structures

Array

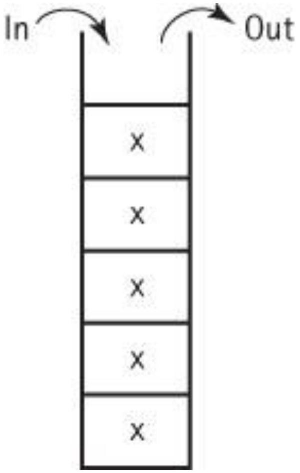


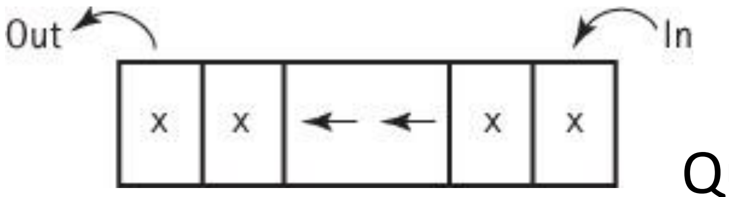
Linked List



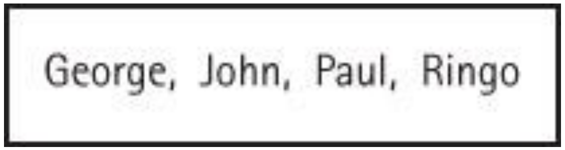
Implementation Independent Structures

Stack



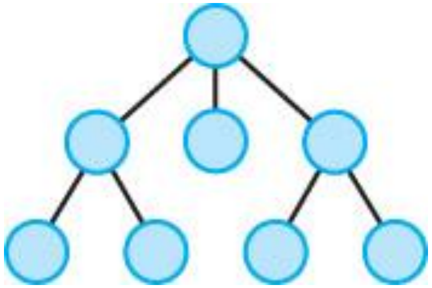


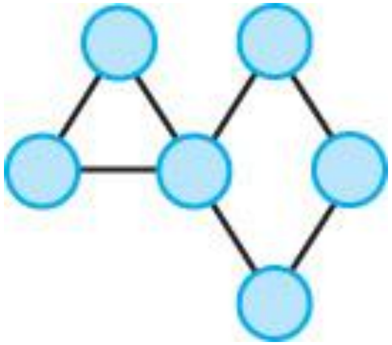
Queue



Sorted List

Tree



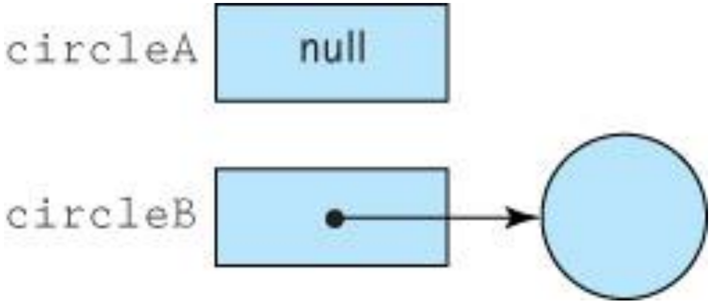


Graph

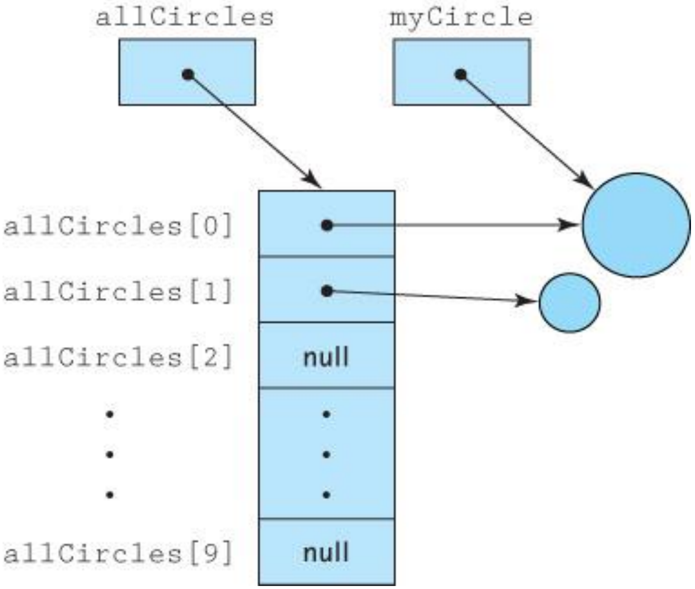
1.6 Basic Structuring Mechanisms

There are two basic structuring mechanisms provided in Java (and many other high level languages)

References



Arrays



Data Structure

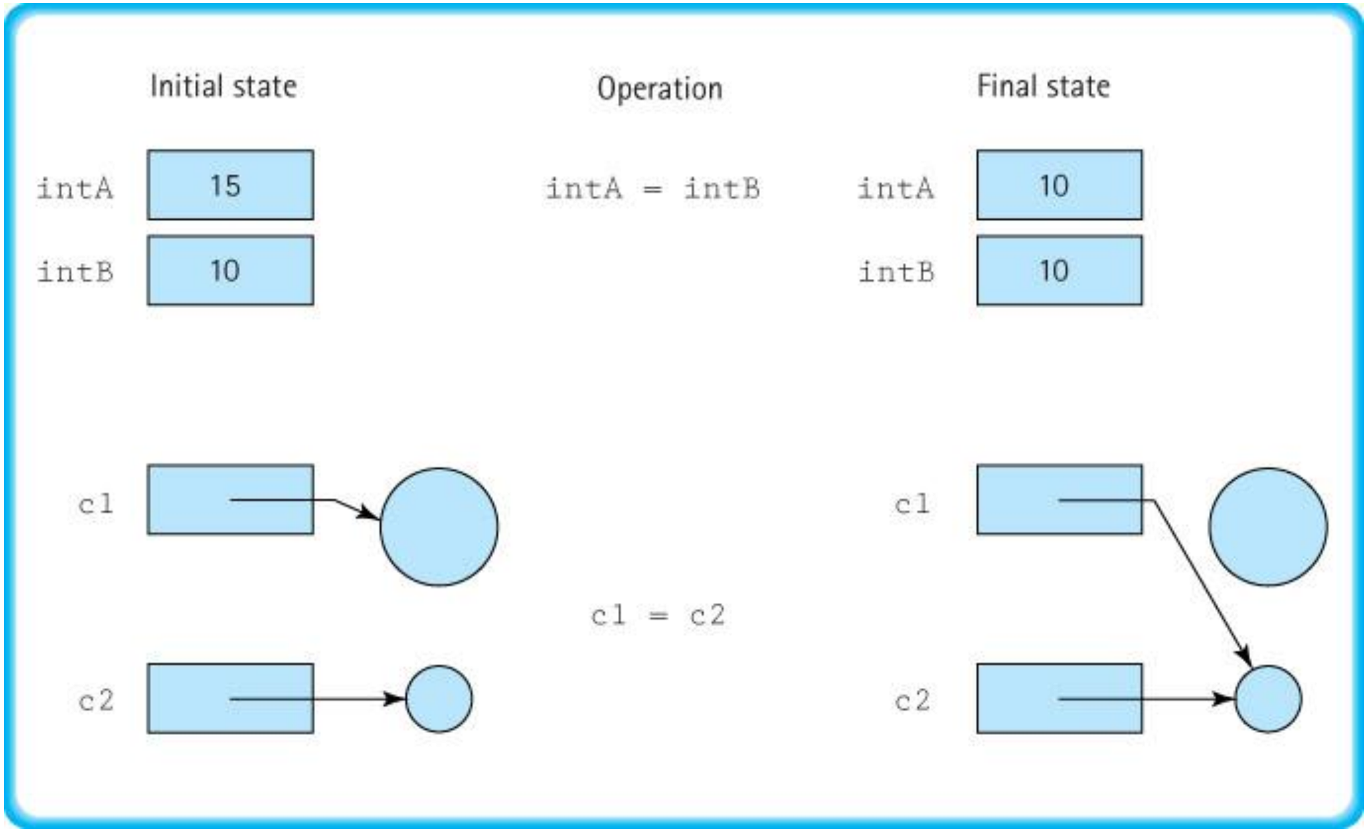
- The implementation of organized data
  - Basically the definition for implementation dependent structures
- Any view of organizing data
  - Inclusive of implementation independent structures

References

- Are memory addresses
- Sometimes referred to as *links*, *addresses*, or *pointers*
- Java uses the reserved word **`null`** to indicate an "absence of reference"
- A variable of a reference (non-primitive) type holds the address of the memory location that holds the value of the variable, rather than the value itself.
- This has several ramifications …

Assignment Statements

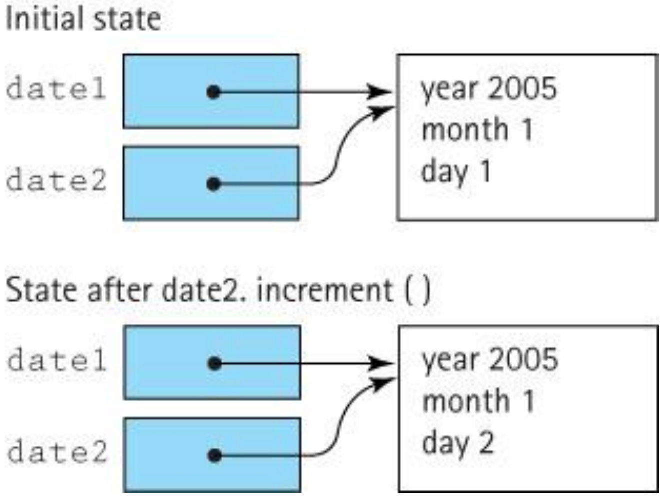
- Reference types: *by reference*
- Primitive types: *by value* (copies value)



Aliases

- The forgetful programmer

```
… // date1 = February, 6th, 2012
date1 = date2;
System.out.println(date1);
date2.increment();
System.out.println(date1);
```

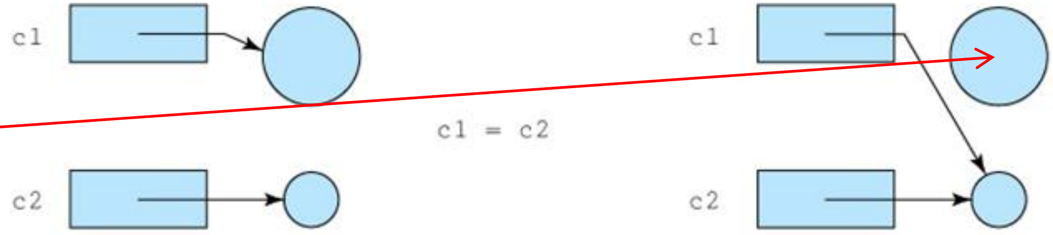


The first println outputs: February, 6th, 2012

What about the second? February 7th, 2012

Garbage – Dynamic Memory Management

- C++ has an added element – a destructor
  - A constructor is used to instantiate an object
    - Thereby, memory for that object is allocated
  - A destructor must be used to destroy the object
    - Thereby, freeing the memory for other applications
    - This is static memory management
- Java has no destructor and manages memory dynamically.

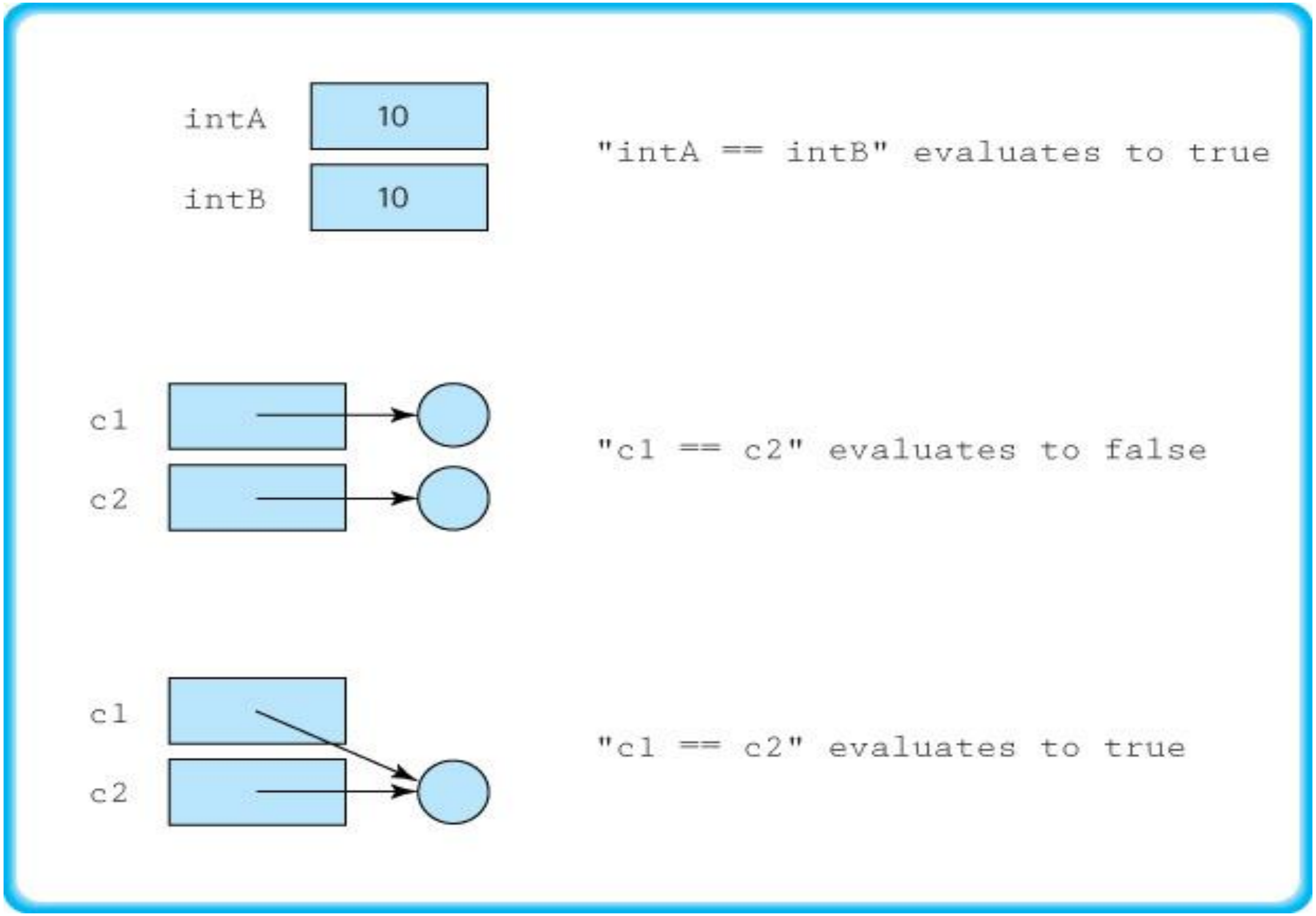


Automatic memory deallocation

Garbage Management

- **Garbage** The set of currently unreachable objects
- **Garbage collection** The process of finding all unreachable objects and deallocating their storage space
- **Deallocate** To return the storage space for an object to the pool of free memory so that it can be reallocated to new objects
- **Dynamic memory management** The allocation and deallocation of storage space as needed while an application is executing

Comparing Objects


intA
10
intB
10
"intA == intB" evaluates to true
c1
c2
"c1 == c2" evaluates to false
c1
c2
"c1 == c2" evaluates to true

Passing Parameters

- In Java, variables are always passed by value
  - A primitive type such as an int is passed as an int
  - A reference type passes the memory address of the object or the array
    - The receiving method must create an alias of the object
    - On return, the original variable finds the object in its modified state, not the original

The Array

- Can use primitive and non-primitive data types
- Specify the size at instantiation

  ```
  int[] numbers = new int[10];
  ```

  all array elements are initialized to null

- Providing initial values for an array is allowed

  ```
  int numbers[] = {1,2,3,4,5,6,7,8,9,10};
  ```

  numbers[0] = 1, …numbers[9] = 10

- What is numbers[10] ?
  - ArrayIndexOutOfBoundsException
  - numbers.length
    - Length is a public instance variable for arrays…
    - In this case, numbers.length would have a value of 10

The Array Continued

- Arrays can specify a collection of objects

  Circle[] allCircles = new Circle[10];

- Two-Dimensional Arrays
  - Table (rows and columns)

  Double [] [] alpha;

  alpha = new double[100,9];

  alpha[0][5] = 36.4

  numCols = alpha.length;

  numRows = alpha[0].length;

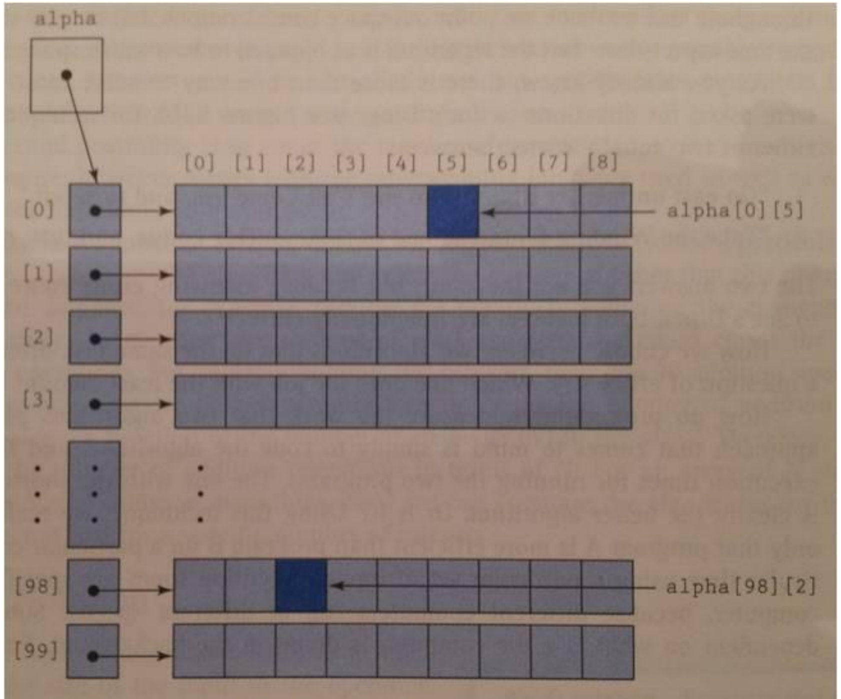


Figure 1.9 *Java implementation of the* alpha *array*

Big-O: Comparison of Growth Rates

N	$\log_2 N$	$N\log_2 N$	N^2	N^3	2^N
1	0	1	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
16	4	64	256	4,096	65,536
64	6	384	4,096	262,144	**requires 20 digits**
128	7	896	16,384	2,097,152	**requires 39 digits**
256	8	2,048	65,536	16,777,216	**requires 78 digits**

Three Complexity Cases

- **Best case complexity** Related to the minimum number of steps required by an algorithm, given an ideal set of input values in terms of efficiency
- **Average case complexity** Related to the average number of steps required by an algorithm, calculated across all possible sets of input values
- **Worst case complexity** Related to the maximum number of steps required by an algorithm, given the worst possible set of input values in terms of efficiency
- To simplify analysis yet still provide a useful approach, we usually use **worst case complexity**

Ways to simplify analysis of algorithms

- Consider worst case only
  - but average case can also be important
- Count a fundamental operation
  - careful; make sure it is the most used operation within the algorithm
- Use Big-O complexity
  - especially when interested in “large” problems

The Big-O
Summing Consecutive Integers

- Sum the integers 1-100
  - Using a control structure

What is the Big-O

Code it

```
int i;
int sum = 0;
int n = 100;
for i = 1; i<=n; i++;
   sum = sum + i;
```

O(N)

  - Using series theory

Validate

sum = (n*(n+1))/2

O(1)

The Big-O
Phone book

• Forward search	$O(N)$
• Reverse search	$O(N)$
• Even split	$O(\log_2 N)$
• Alphabetical	$O(\log_{26} N)$

CIS 113 Downloads

- Install the bookFiles on your thumbdrive
  - Copy the source code from CIS113 on myAVC

Chapter 1
Group Exercises and Homework

- **Chapter 1 exercises.**
  - 2, 11, 17, 20, 28, 39, 46, 48, 51
  - Programming: 26 b
  - **Extra credit**: combine: 22, 23a and 23b
  - E-mail me your homework (avc.dr.lee@gmail.com)

- Be prepared for a short quiz on …
  - the Big-O notation