# CSC 1052 – Algorithms & Data Structures II:

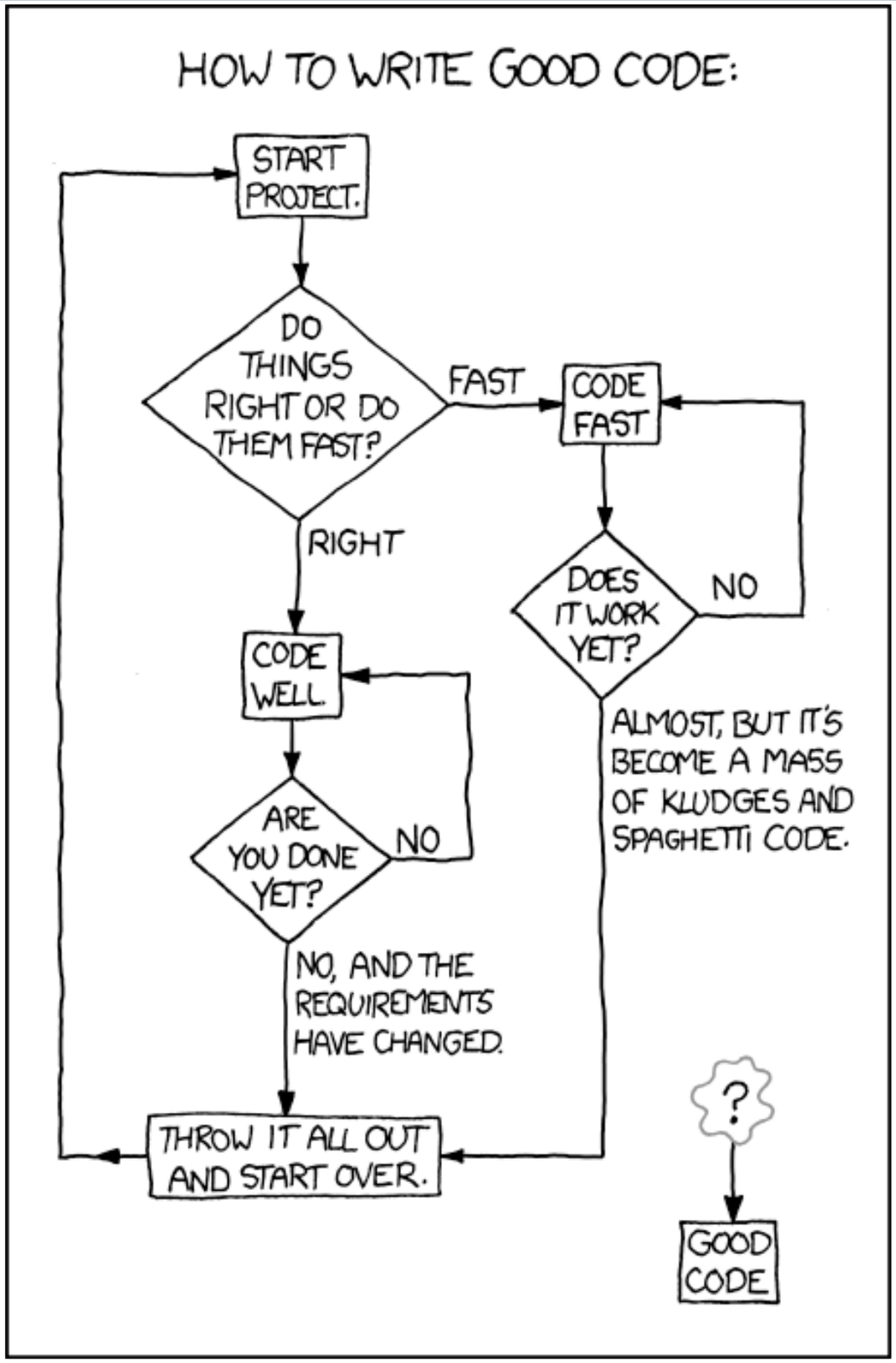
# Java Constructions

Professor Henry Carter
Spring 2017

# Recap

- Java basics
  - Primitive types
  - Control flow
  - Classes and objects
- Java compilation
- Organize our approach and abstract our thinking

# Code vs. Good Code



# Code vs. Good Code

- Organize your code
- Provide helpful error messages
- Comment your code

# Organizing Classes

- Recall: classes define the variables and methods in an object
- Create separate classes for objects with different attributes and responsibilities
- What if these attributes and responsibilities overlap?

# Inheritance

- Allows creating a new class that augments the functionality of an existing class
- Maintains organizational separation while re-using critical code
- The new class is the *subclass*, while the existing class is the *superclass* of the new class

# Example

```
package ch01.dates;
public class Date
{
  protected int year, month, day;
  public static final
      int MINYEAR = 1583;

  // Constructor
  public Date(int newMonth,
              int newDay,
              int newYear)
  {
    month = newMonth;
    day = newDay;
    year = newYear;
   }

  // Observers
  public int getYear()
  {
    return year;
  }

  public int getMonth()
  {
    return month;
  }

  public int getDay()
  {
    return day;
  }

  public int lilian()
  {
    // Returns the Lilian Day Number
    // of this date.
    // Algorithm goes here.
  }

  @Override
  public String toString()
  // Returns this date as a String.
  {
    return(month + "/" + day
                 + "/" + year);
  }
}
```

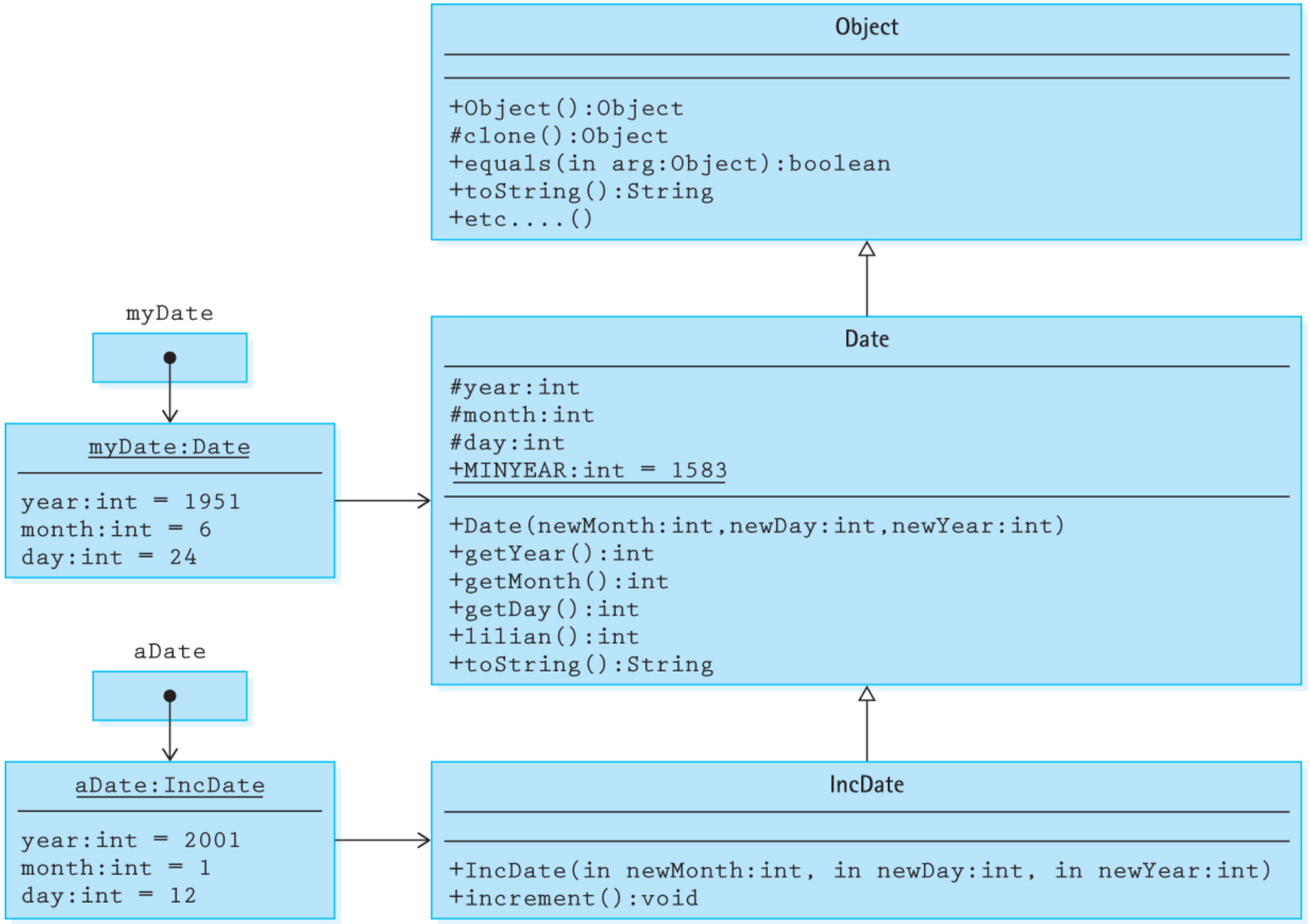
# Example

```
package ch01.dates;

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

  public void increment()
  // Increments this IncDate to represent the next day.
  // For example if this = 6/30/2005 then this becomes 7/1/2005.
  {
    // increment algorithm goes here
  }
}
```

# Example

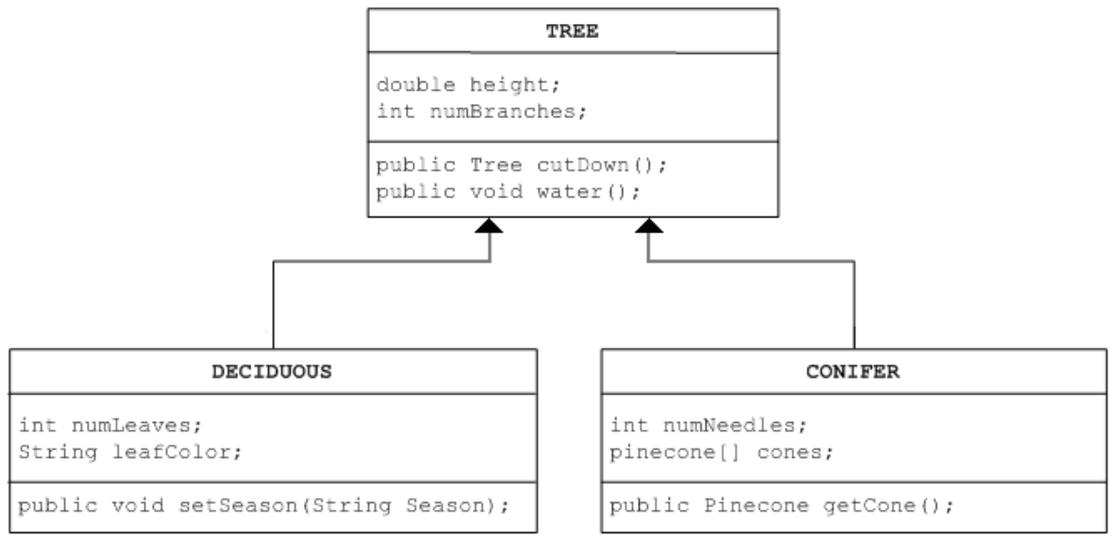
```
Date myDate = new Date(6, 24, 1951);
IncDate aDate = new IncDate(1, 11, 2001);

System.out.println("mydate day is: " + myDate.getDay());
System.out.println("aDate day is:  " + aDate.getDay());

aDate.increment();

System.out.println("the day after is: " + aDate.getDay());
```

# UML Diagram


Object
+Object():Object
#clone():Object
+equals(in arg:Object):boolean
+toString():String
+etc....()
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
IncDate
+IncDate(in newMonth:int, in newDay:int, in newYear:int)
+increment():void
myDate
myDate:Date
year:int = 1951
month:int = 6
day:int = 24
aDate
aDate:IncDate
year:int = 2001
month:int = 1
day:int = 12

# Inheritance Tree

- Java supports single inheritance only.
- Method calls are resolved by searching *up* the inheritance tree.
- All Java classes can be traced up to the Object class.
- We use the `@Override` notation to indicate the redefinition of an inherited method.



# Inheritance-Based Polymorphism

- Strong vs weak typing
- Polymorphism – an object variable can reference objects of *different* classes at *different* times.
- A variable declared to be of "type" `T` can hold a reference to an object of type `T` or of any descendant of `T`.

# Example

```
// cutoff is random int between 1 and 100
Object obj;
if (cutoff <= 50)
    obj = new String("Hello");
else
    obj = new Date(1,1,2015);
System.out.println(obj.toString());
```

# Practice

```
int temp;
Date date1 = new Date(10, 2, 1989);
IncDate date2 = new IncDate(12,25,2001);
…
```

```
temp = date1.getDay();
temp = date2.getDay();
date1.increment();
date2.increment();
date2 = new Date(1,19,2017);
```

# Packaging it up

- Java groups related classes into a *package*
- The keyword *package* followed by an identifier indicating the name of the package:
  - `package someName;`
- Import declarations, to make the contents of other packages available:
  - `import java.util.Scanner;`
  - `import java.util.*;`

# Exceptional Situations

- Erroneous program behavior can occur for a plethora of reasons
- Java exception objects allow developers to identify problems spots and ensure they are handled correctly
- Three components:
  - Define the exception as a subclass of the `Exception` class
  - Generate the exception (throw it)
  - Handle the exception with a `try-catch` block (catch it)

# Example

```
public class DateOutOfBoundsException extends Exception
{
  public DateOutOfBoundsException()
  {
    super();
  }

  public DateOutOfBoundsException(String message)
  {
    super(message);
  }
}
```

# Example

```
public SafeDate(int newMonth, int newDay, int newYear)
           throws DateOutOfBoundsException
{
  if ((newMonth <= 0) || (newMonth > 12))
    throw new DateOutOfBoundsException("month " + newMonth + "out of range");
  else
    month = newMonth;

 day = newDay;

 if (newYear < MINYEAR)
    throw new DateOutOfBoundsException("year " + newYear +
                                       " is too early");
 else
    year = newYear;
}
```

# Example

```
public class UseSafeDate
{
  public static void main(String[] args)
                          throws DateOutOfBoundsException
  {
    SafeDate theDate;
    // Program prompts user for a date
    // M is set equal to user's month
    // D is set equal to user's day
    // Y is set equal to user's year
    theDate = new SafeDate(M, D, Y);

    // Program continues ...
  }
}
```

RESULTS:

```
Exception in thread "main" DateOutOfBoundsException: year 1051 is too early
        at SafeDate.<init>(SafeDate.java:18)
        at UseSafeDate.main(UseSafeDate.java:57)
```

# Example

```
public class UseSafeDate
{
  public static void main(String[] args)
  {
    SafeDate theDate;
    boolean DateOK = false;

    while (!DateOK)
    {
      // Program prompts user for a date
      // M is set equal to user's month,
      // D is set equal to user's day
      // Y is set equal to user's year
      try
      {
        theDate = new SafeDate(M, D, Y);
        DateOK = true;
      }
      catch(DateOutOfBoundsException DateOBExcept)
      {
        output.println(DateOBExcept.getMessage());
      }
    }

    // Program continues ...
  }
}
```

# General Rules

- An exception may be handled any place in the software hierarchy
- Unhandled built-in exceptions carry the penalty of program termination
- Exceptions should always be handled at a level that knows what the exception means
- An exception need not be fatal
- For non-fatal exceptions, execution should continue from the lowest level that can recover from the exception.

# //Comments Section

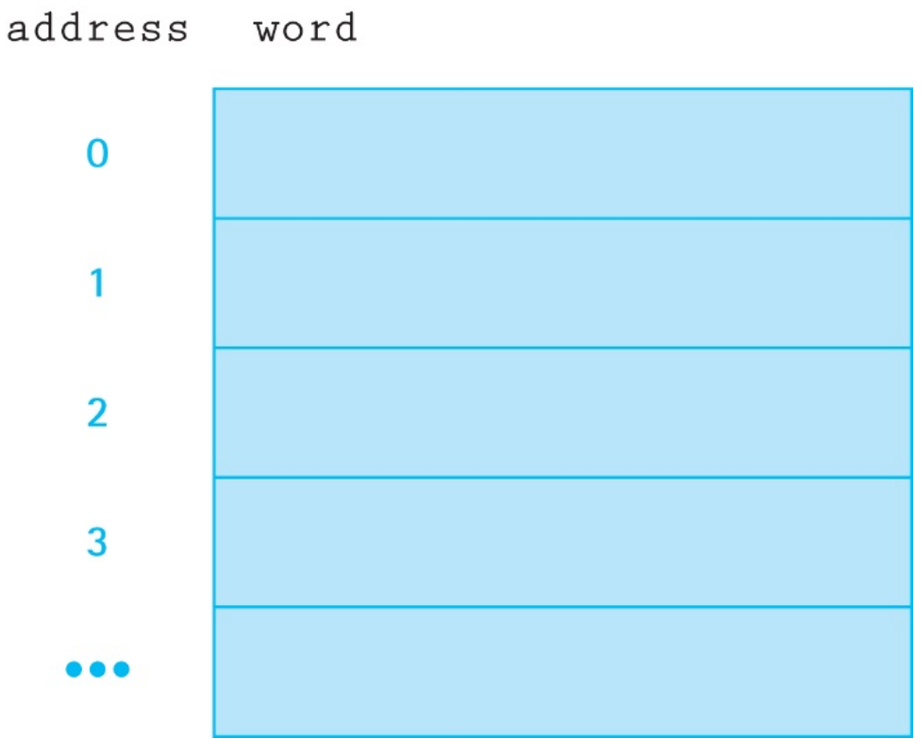
- Helpful for code review and reuse (and grading)
- Java provides three types of comments:
  - Line (//This is a comment)
  - Block  (/* This comment could span many lines */)
  - Javadoc comment (/** This comment can be used to generate an HTML guide **/)
- Practice commenting your code now! This will be part of your project and lab grades!

# Looking Ahead: Data Structures

- The way you arrange and access data is critical to simplifying your programs and making them efficient
- The primitive types used in Java are not enough to represent all possible types and arrangements of information
- Data structures provide this ability

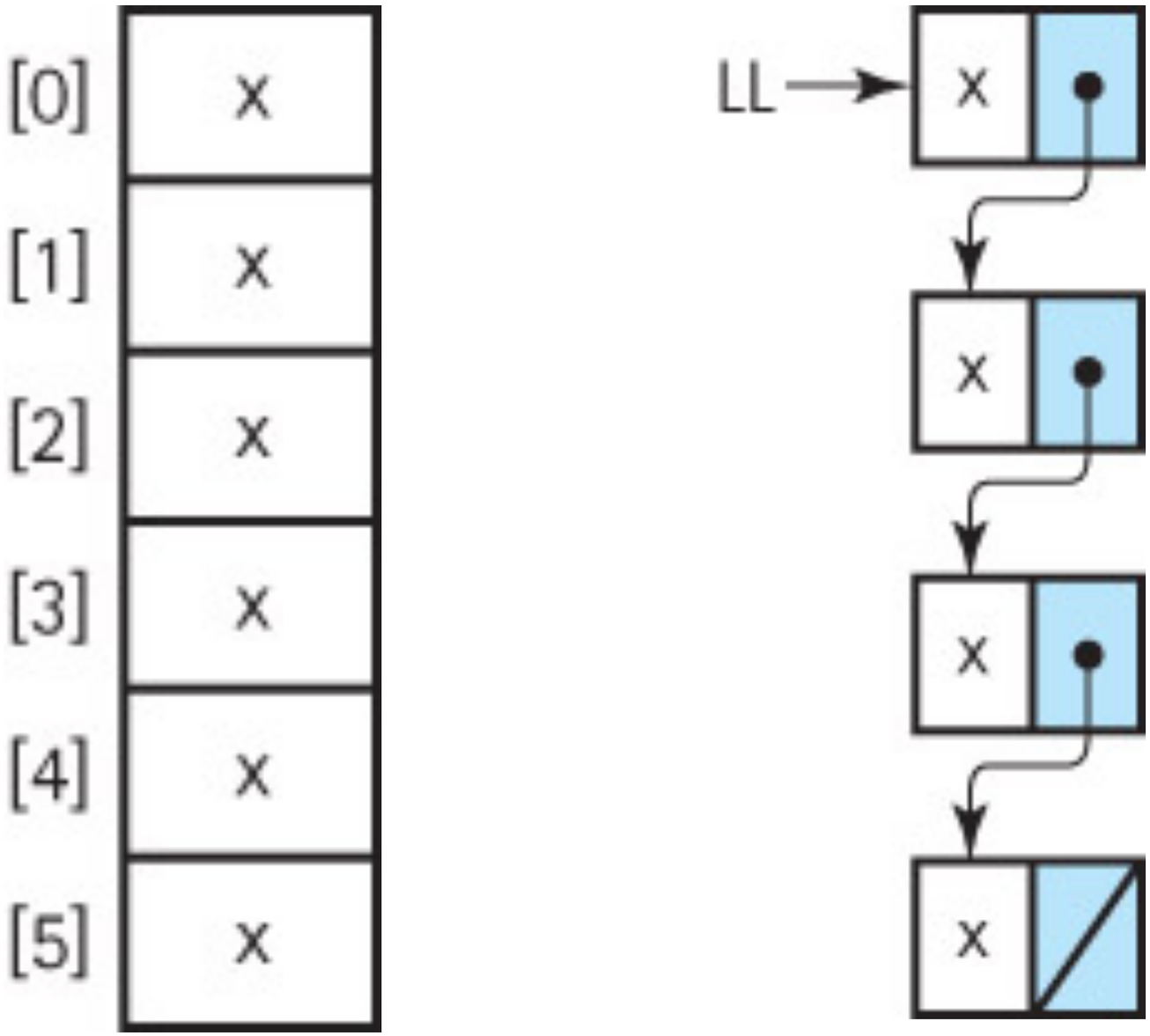
# Memory

- All programs and data are held in memory.
- Memory consists of a contiguous sequence of addressable words.
- A variable in our program corresponds to a memory location.



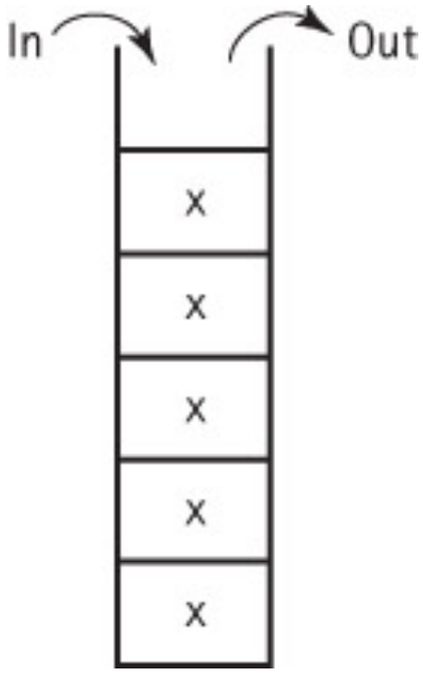
# Implementation Dependent

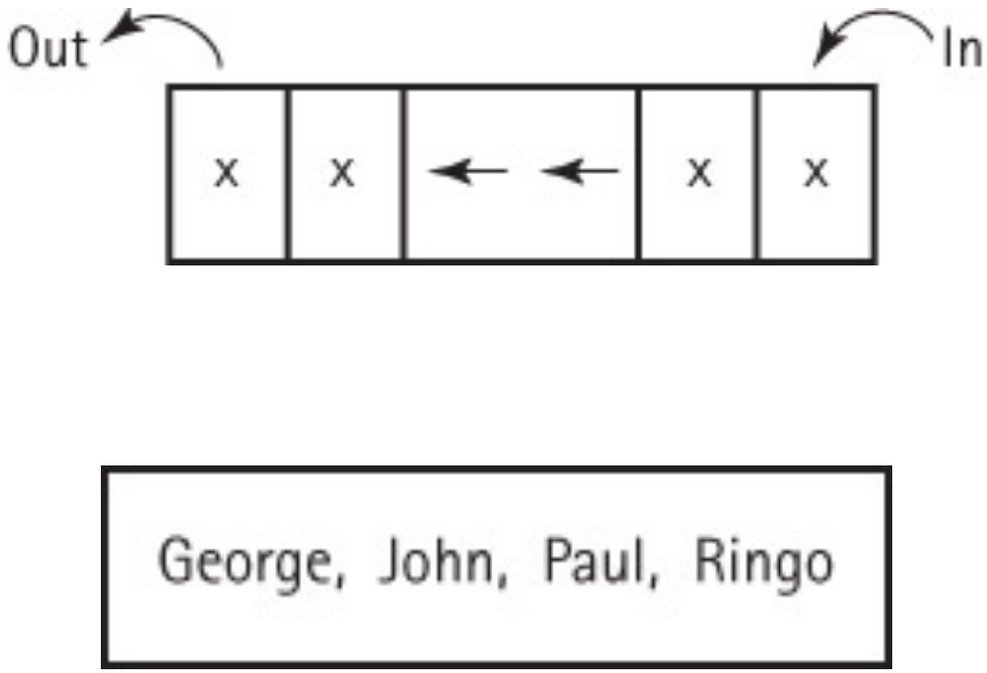
- Arrays vs. Linked Lists

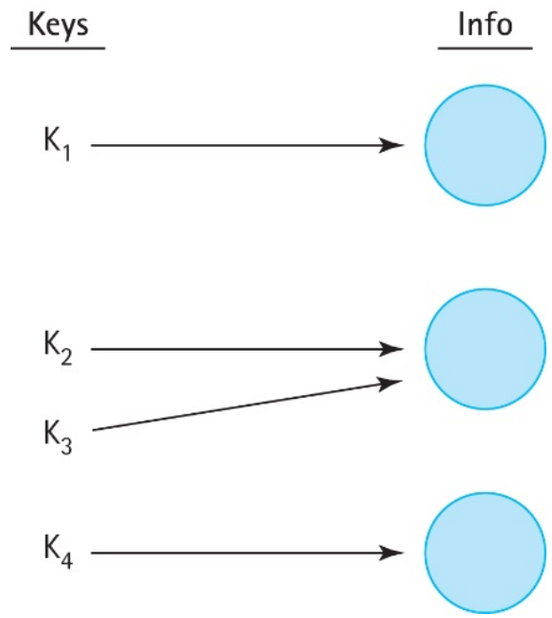


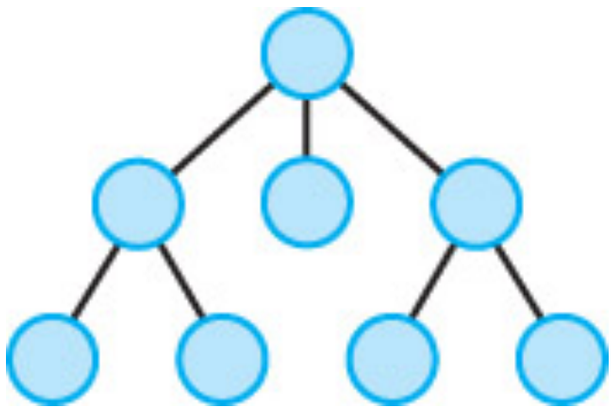
# Implementation Independent

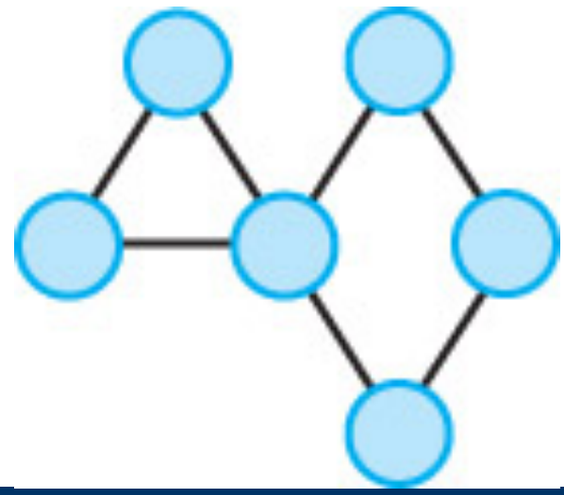
- Behavior-based



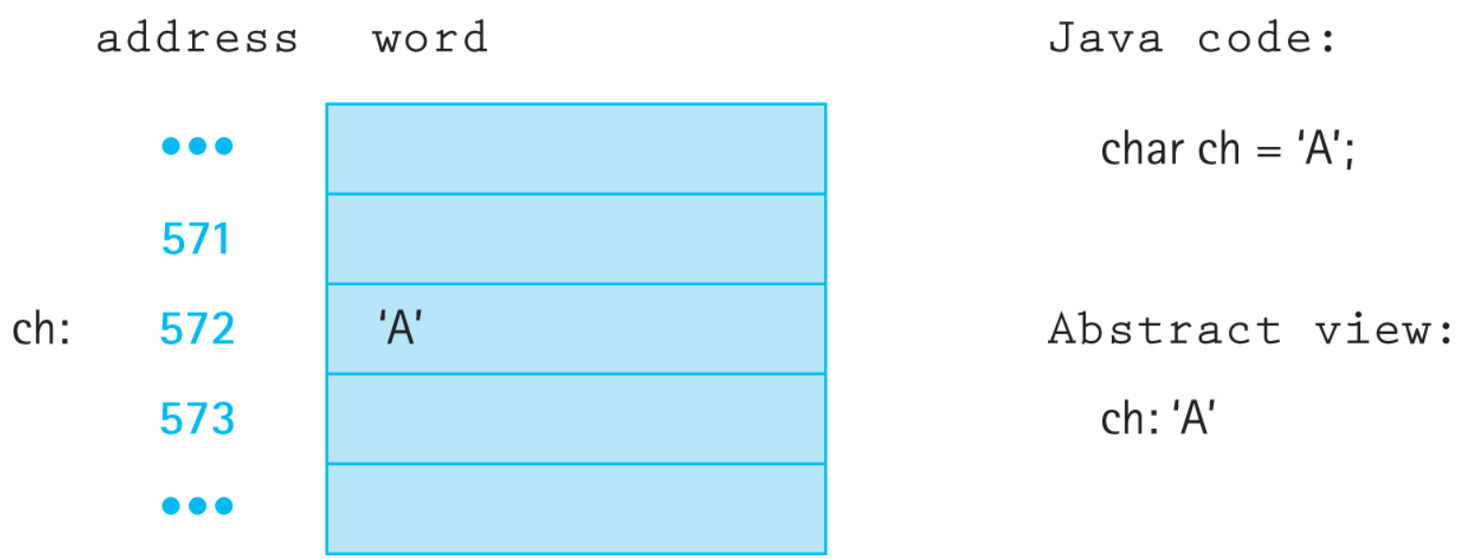





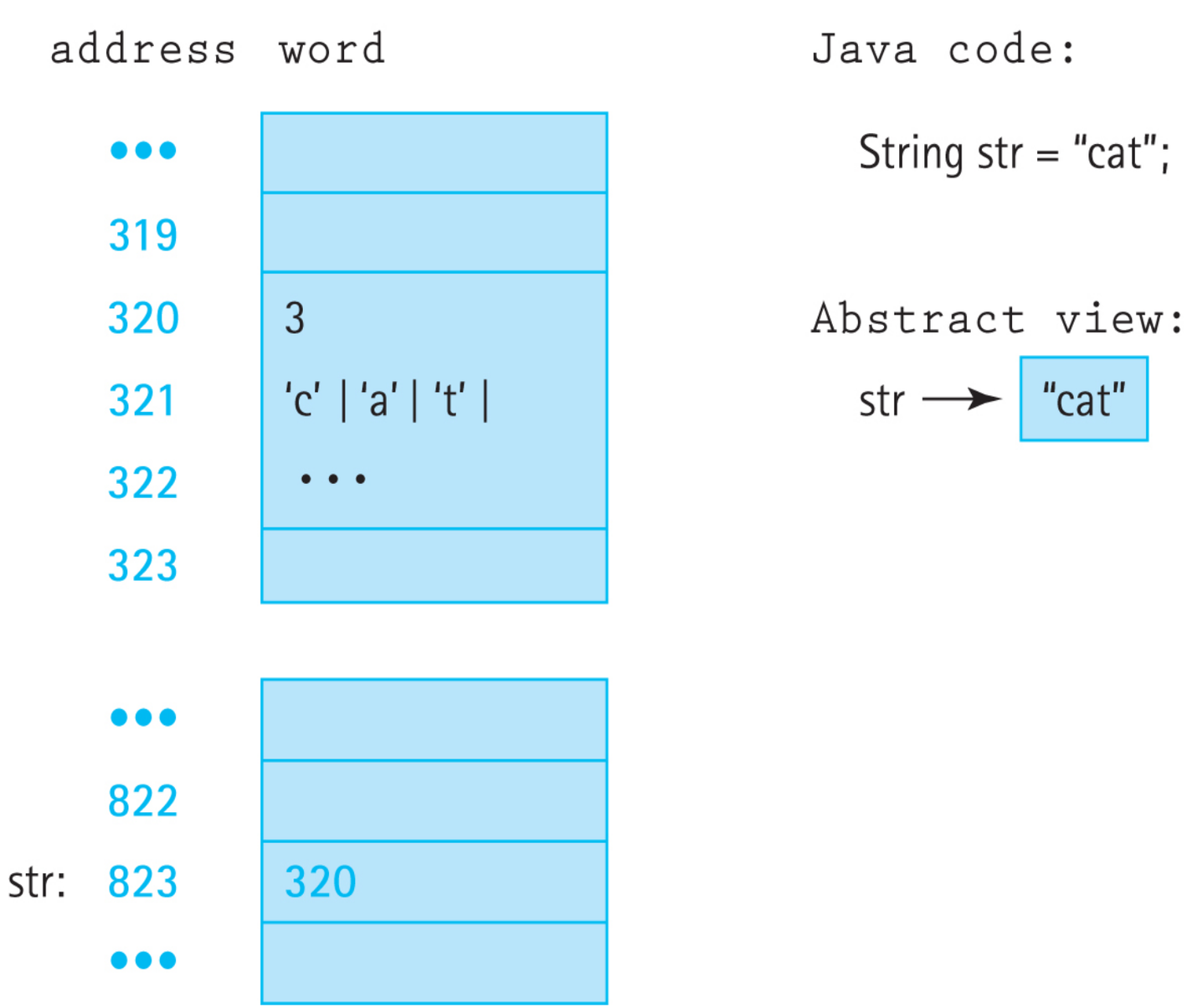
# Direct vs Indirect Addressing

- Direct:



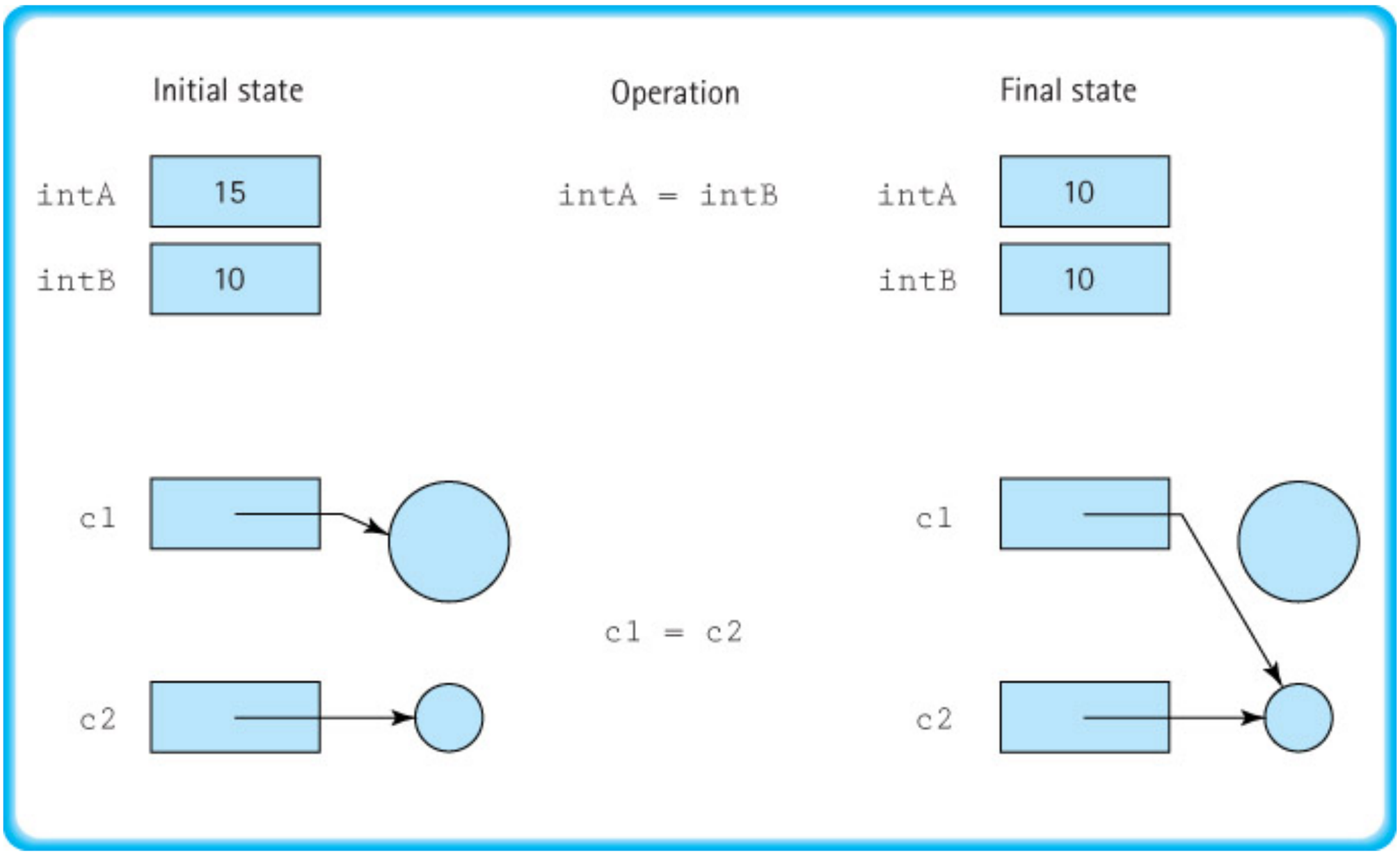
# Direct vs Indirect Addressing

- Indirect:



# Assigning values

- The " = " operator



# Practice

```
int i = 10;
int j = 20;
String str1 = “cat”;
…
int j = i;
String str2 = str1;
```

# Practice

```
date1 = new IncDate(1,19,2017);
date2 = date1;

System.out.println(date1);
System.out.println(date2);
```

```
date1.increment();

System.out.println(date1);
System.out.println(date2);
```

# Some Details

- Comparison
  - Compares the *value* stored in that variable
- Parameter Passing in Methods
  - Java is pass-by-value
    - What does this imply for objects?
- Garbage Collection

# Arrays

- Contiguous allocated memory with offsets
- Primitive array vs. object array
- 2D arrays (matrix)

# Recap

- Good programming requires careful practice
  - Organize your code
  - Throw useful exceptions and errors
  - Comment your code
- Data structures allow us to arrange data for fast and easy access
- The arrangement of data in memory greatly impacts the way Java handles the data

# Next Time...

- Dale, Joyce, Weems Chapter 1.6
  - Remember, you need to read it BEFORE you come to class!
- Check the course webpage for practice problems
- Peer Tutors
  - http://www.csc.villanova.edu/help/