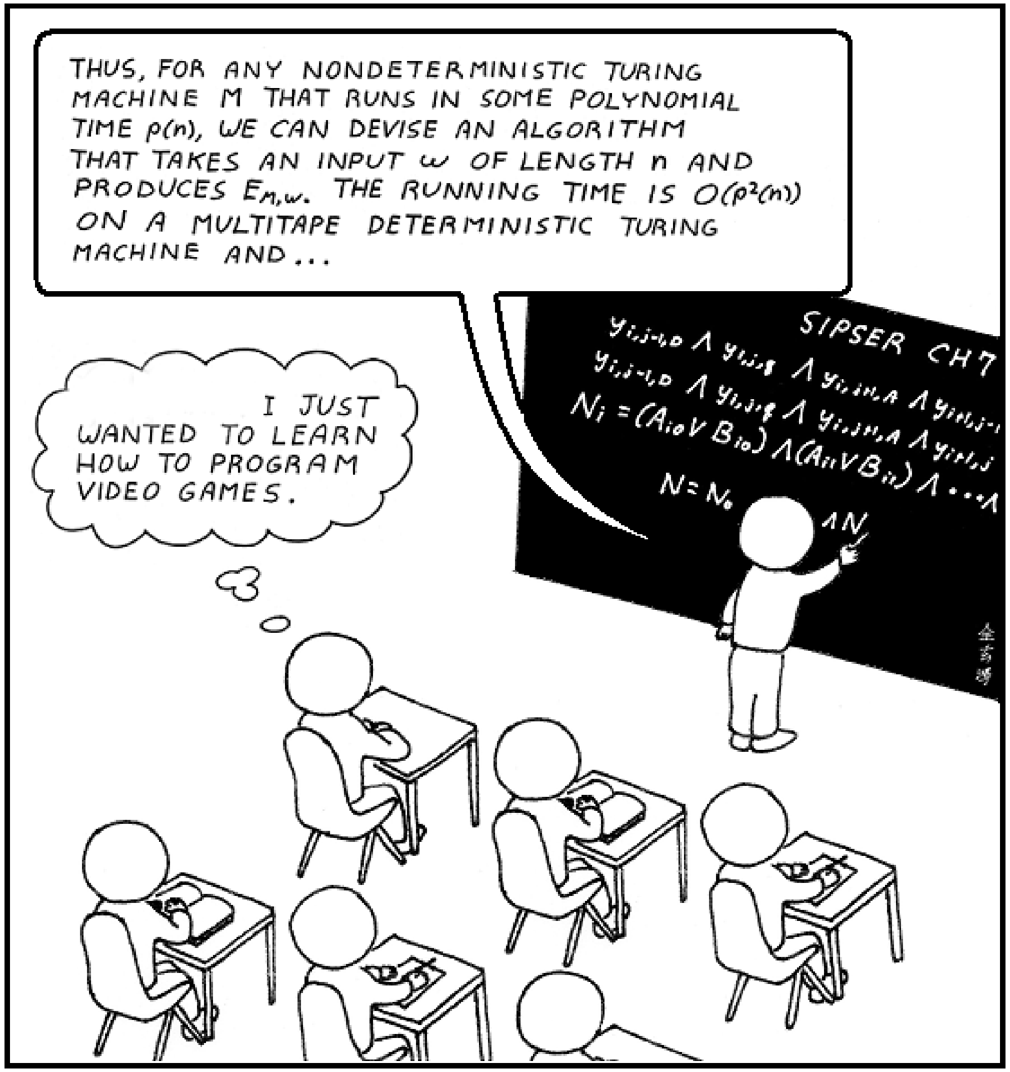
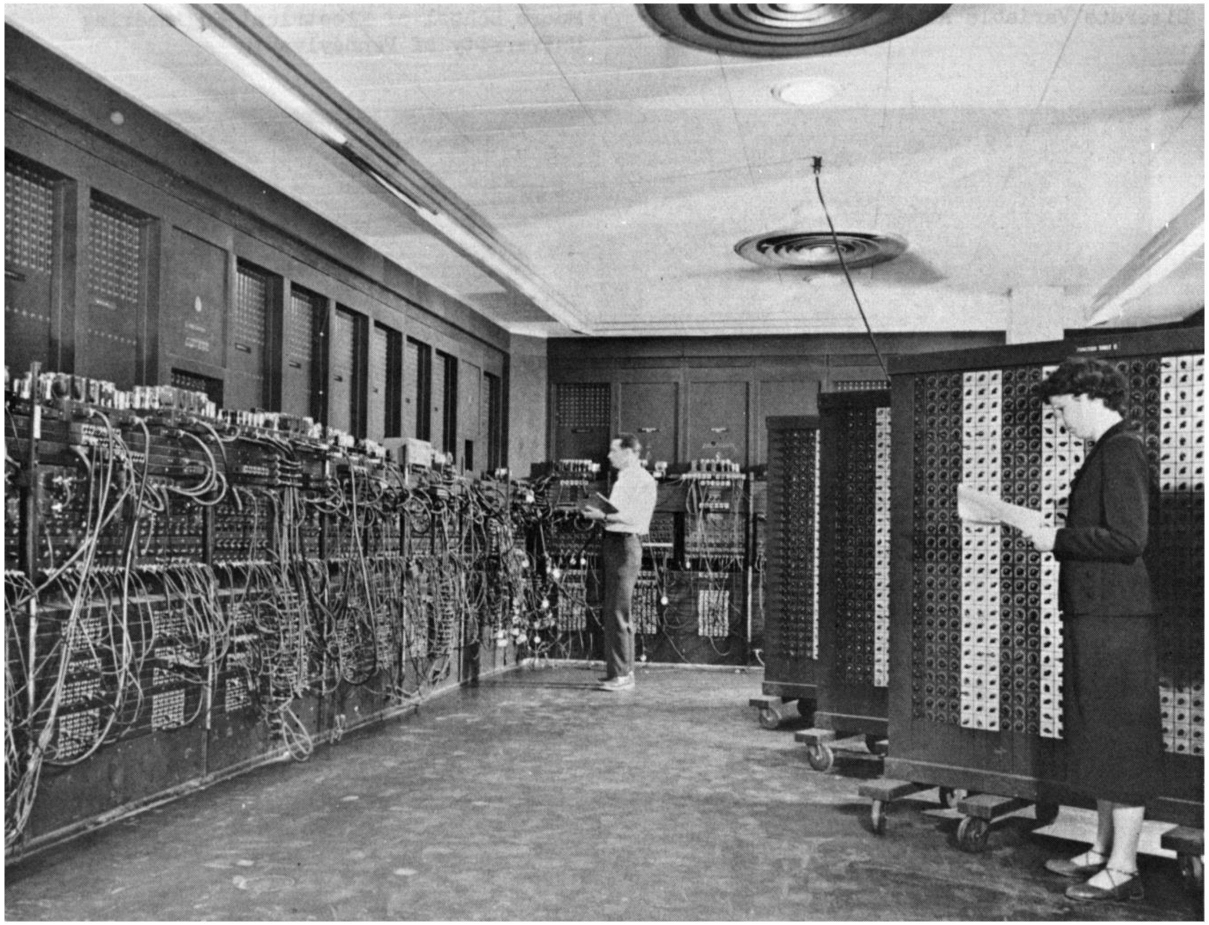
Java Class 2


THUS, FOR ANY NONDETERMINISTIC TURING MACHINE M THAT RUNS IN SOME POLYNOMIAL TIME p(n), WE CAN DEVISE AN ALGORITHM THAT TAKES AN INPUT ω OF LENGTH n AND PRODUCES E_{M,w}. THE RUNNING TIME IS O(p²(n)) ON A MULTITAPE DETERMINISTIC TURING MACHINE AND...
I JUST WANTED TO LEARN HOW TO PROGRAM VIDEO GAMES.
SIPSER CH7

ENIAC Computer



ENIAC Computer

The **ENIAC** was designed to calculate artillery firing tables for the United States Army. When ENIAC was announced in 1946 it was heralded as a "Giant Brain.“ It had a speed of one thousand times that of electro-mechanical devices. When fully operational, ENIAC occupied a room that was 30 by 50 feet in size and weighed 30 tons. A total of 40 panels were arranged in a U-shape that measured 80 feet long at the front and contained over 18,000 vacuum tubes. When you “programmed” the ENIAC you punched cards and connected cables. You wrote programs in machine language – telling the computer step-by-step what you wanted it to do.

Computers and Languages

The genesis of the computer revolution was in a machine

The genesis of our programming languages looked like that machine

Steve Jobs: computers are “bicycles for the mind”

Computer tools now are looking less like machines and more like parts of our minds

Language Levels

Each type of CPU has its own specific ***machine language***

The other levels were created to make it easier for a human being to read and write programs

There are a number of programming language levels:

- ***machine language: 01100001 (1st generation)***
- ***assembly language: LDA 1 (2nd generation)***
- ***high-level language: X=1; (3rd generation)***
- ***languages to access databases (4th generation)***
- ***Languages for artificial intelligence and neural nets (5th generation)***

Programming Languages

A program must be translated into machine language before it can be executed

A ***compiler*** is a software tool which translates source code into a specific target language

Often, that target language is the machine language for a particular CPU type

The Java approach is somewhat different

Java Translation

The Java compiler translates Java source code into a special representation called ***bytecode***

Java bytecode is not the machine language for any traditional CPU

Another software tool, called an ***interpreter***, translates bytecode into machine language and executes it statement by statement

Therefore the Java compiler is not tied to any particular machine

Java is considered to be architecture-neutral

Compiler vs. Interpreter

Compiler:

English

Java is a programming language.

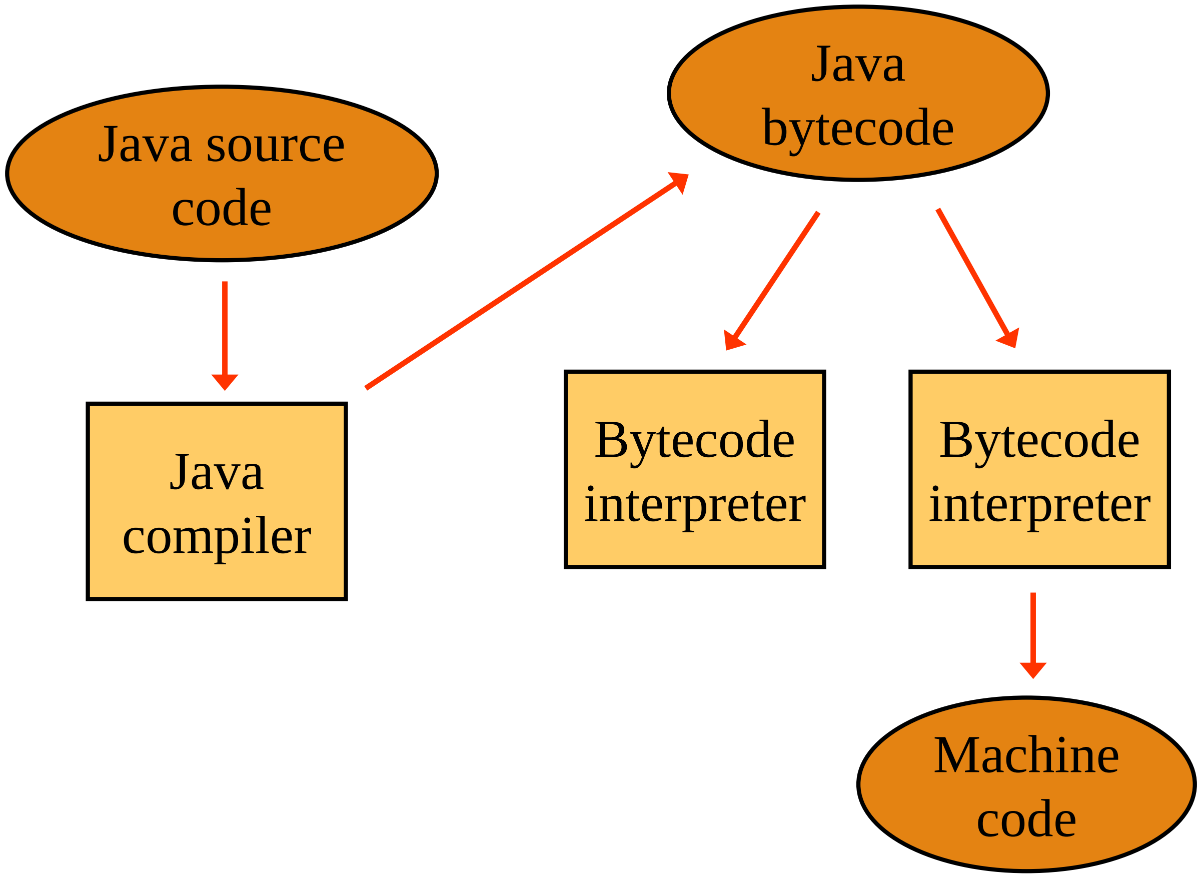
Spanish

Java es un lenguaje de programación.

Interpreter:

English		*Spanish*
Java	→	Java
is	→	es
a	→	un
programming	→	de programación
language	→	lenguaje

Java Translation



Quick Check

Select the word from the following that best matches each of the following:

Machine, assembly, compiler, high-level, IDE (Integrated Development Environment), interpreter

A program written in this type of language can run directly on a computer.

machine

Generally, each language instruction in this type of language corresponds to an equivalent machine language instruction.

assembly

Most programmers write their programs using this type of language processor

IDE

Java is an example of this type of language.

high-level

This type of program translates code in one language to code in another language.

compiler

This type of program interweaves the translation of code and the execution of code.

interpreter

Identifiers Revisited

Identifiers are the "words" in a program

- Name of a class: `Dog`, `MyFirstApp`, and `Lincoln`
- Name of a method: `bark`
- Name of a variable: `total`

Review of rules for identifiers:

- A Java identifier can be made up of letters, digits, the underscore character (_), and the dollar sign ($)
- Identifiers cannot begin with a digit
- Java is case sensitive: `Total`, `total`, and `TOTAL` are different identifiers
- Don't use reserved words for your own names

Identifiers Revisited

By convention, programmers use different case styles for different types of identifiers, such as

- *title case* for simple class names – `Lincoln`
- *camel case* for compound words – `HelloWorld, totTax`
- *upper case* for constant variables – `MAXIMUM`

How about the identifiers in

```
System.out.println ("Whatever you are, be a good one.");
```

`System`, `out`, and `println` ?

- Not part of the Java language
- Part of the ***Java standard library*** – a set of predefined classes and methods that someone (another programmer) has already written for us

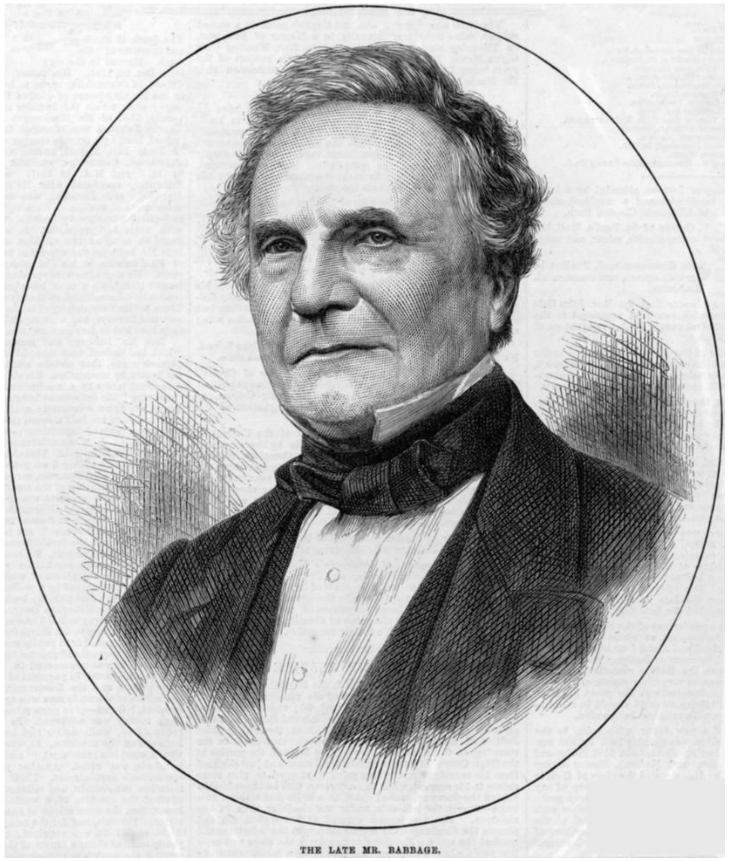
Syntax and Semantics

The ***syntax rules*** of a language define how we can put together symbols, reserved words, and identifiers to make a valid program

The ***semantics*** of a program statement define what that statement means (its purpose or role in a program)

A program that is syntactically correct is not necessarily logically (semantically) correct

A program will always do what we tell it to do, not what we meant to tell it to do



Charles Babbage, often referred to as "The Father of Computing," invented the first mechanical computer in the early 1800s.

"I have been asked, 'Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?' I am not able to rightly apprehend the kind of confusion of ideas that could provoke such a question." Charles Babbage (1791-1871)

Errors

A program can have three types of errors

The compiler will find syntax errors and other basic problems (*compile-time or syntax errors*)

- If compile-time errors exist, an executable (bytecode) version of the program is not created

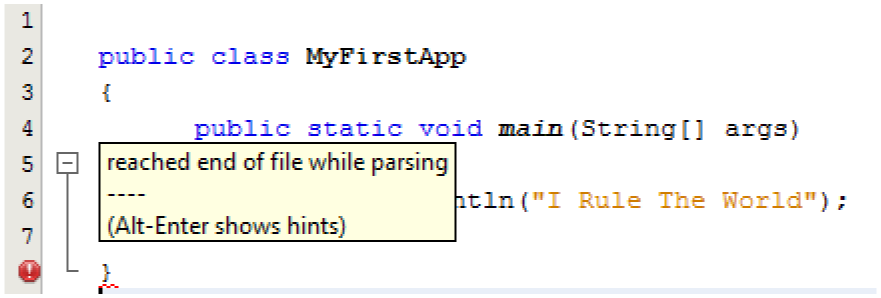
A problem can occur during program execution by the interpreter, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)

A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)

Syntax Errors

Compile-time or syntax errors: the compiler will find syntax errors and other basic problems

```
public class MyFirstApp
{
    public static void main(String[] args)
    {
    System.out.println("I Rule The World!");

}
```



```

public class MyFirstApp
{
        public static void main(String[] args)
reached end of file while parsing
----
(Alt-Enter shows hints)
                                  ntln("I Rule The World");

}
```

Syntax Errors

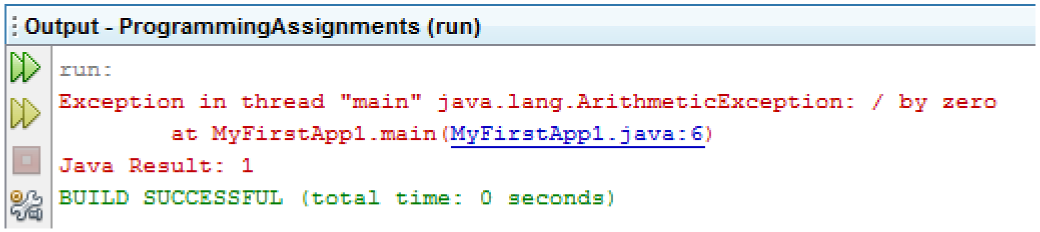
```
public class MyFirstApp
{
    public static void main(String[] args)
    {
    System.out.println("I Rule The World!");
    }
}
```

missing bracket

Run-Time Errors

Run-time errors: a problem can occur during program execution, which causes a program to terminate abnormally

```
public class MyFirstApp1
{
        public static void main(String[] args)
       {
           System.out.println(12/0);
       }
}
```


Output - ProgrammingAssignments (run)

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at MyFirstApp1.main(MyFirstApp1.java:6)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Semantic Errors

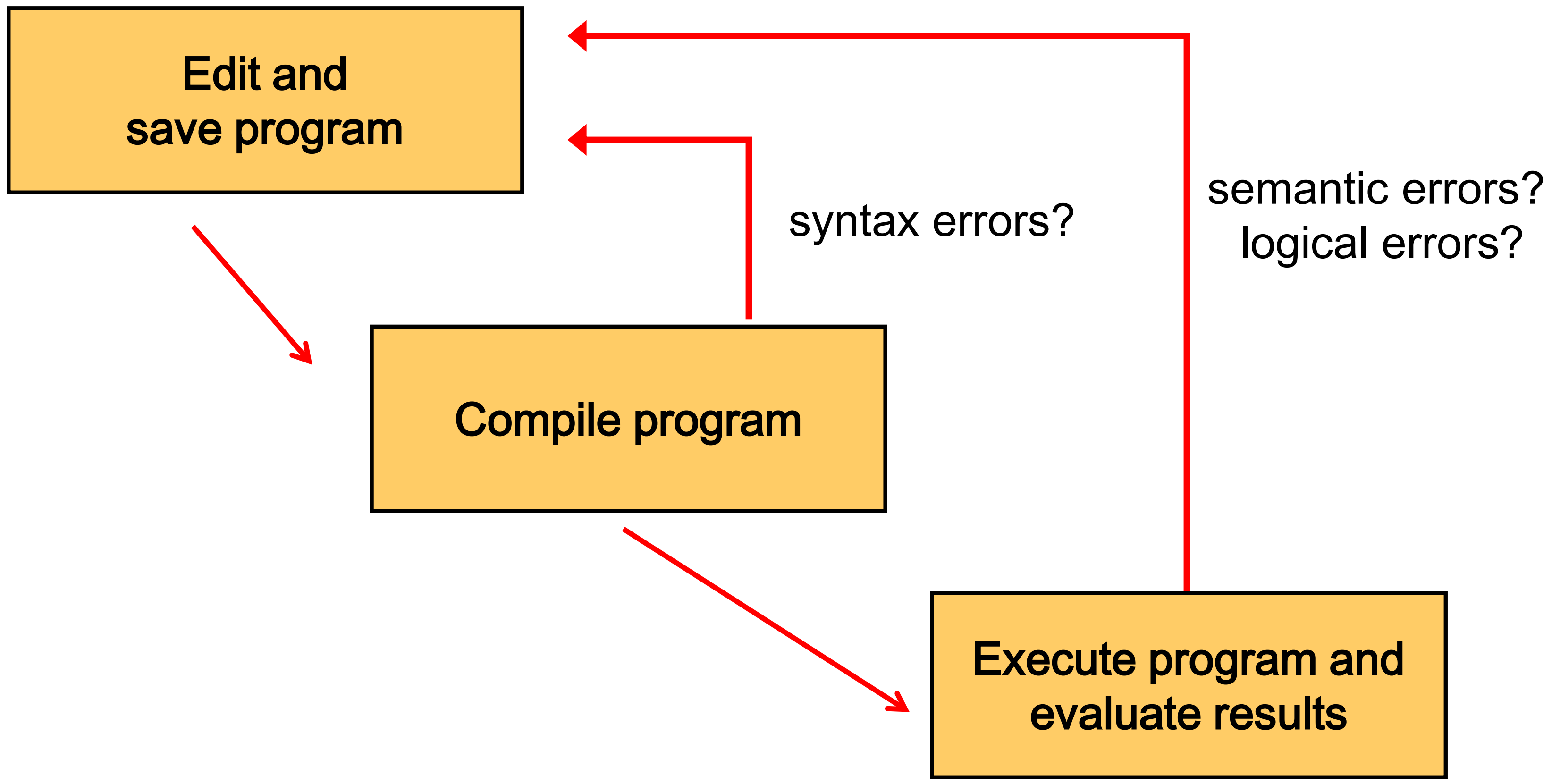
Logical or semantic errors: a program may run, but produce incorrect results, perhaps because of an incorrect formula – remember Charles Babbage

```
 discriminant = Math.pow(b, 2) - (4 * a * c);
     root1 = (b + Math.sqrt(discriminant)) / (2 * a);
     root2 = (b - Math.sqrt(discriminant)) / (2 * a);
```

✗

```
   discriminant = Math.pow(b, 2) - (4 * a * c);
     root1 = ((-1*b) + Math.sqrt(discriminant)) / (2 * a);
     root2 = ((-1*b) - Math.sqrt(discriminant)) / (2 * a);
```

Basic Program Execution



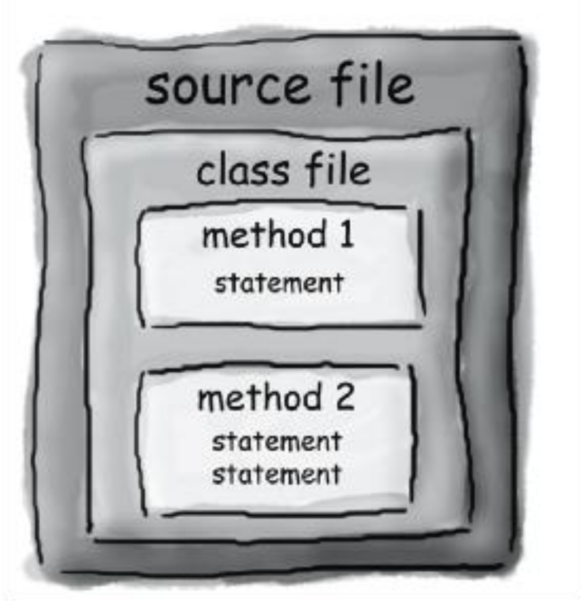
Problem Solving

The purpose of writing a program is to solve a problem

The key to designing a solution is breaking it down into manageable pieces

An ***object-oriented*** *approach* lends itself to this kind of solution decomposition

We will dissect our solutions into pieces called classes (.java files)



Put a class in a source file.

Put methods in a class.

Put statements in a method.

Object-Oriented Programming

Classes are the fundamental building blocks of an objected-oriented program

- Each class is an abstraction of similar objects in the real world, e.g., dogs, songs, houses, etc.

Once a class is established, multiple objects can be created from the same class

- Each object is an instance (a concrete entity) of a corresponding class, e.g., a particular dog, a particular song, a particular house, etc.

Classes

An *object* is defined by a *class*

A *class* is the *blueprint of an object*

The *class* uses *methods* to define the behaviors of the object

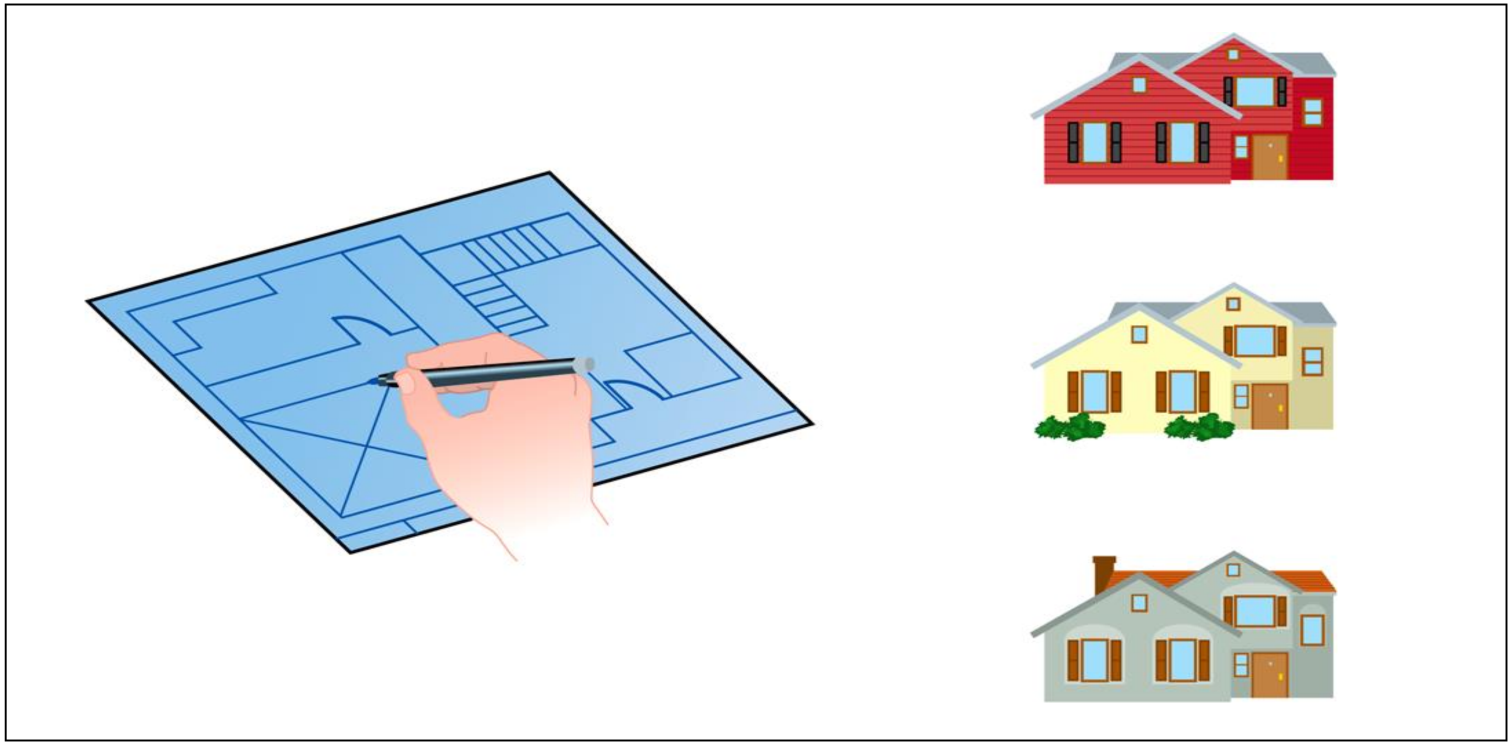
The class that contains the *main method* of a Java program represents the entire program

A *class* represents a *concept*, and an *object* represents the *embodiment of that concept*

Multiple objects can be created from the same class

Class = Blueprint

One blueprint to create several similar, but different, houses:

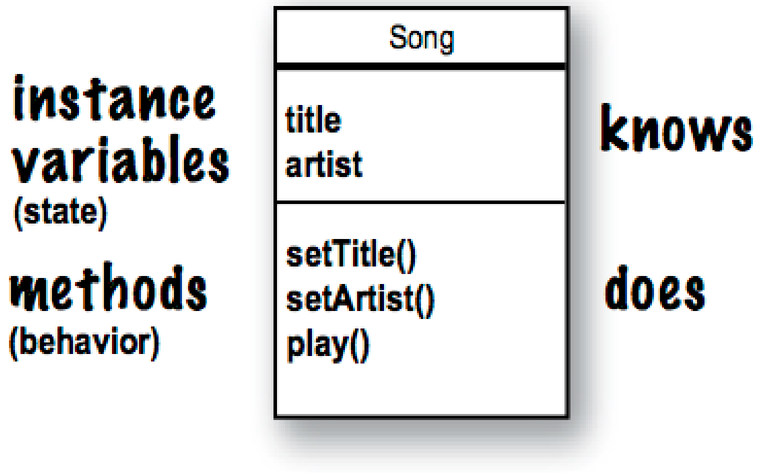


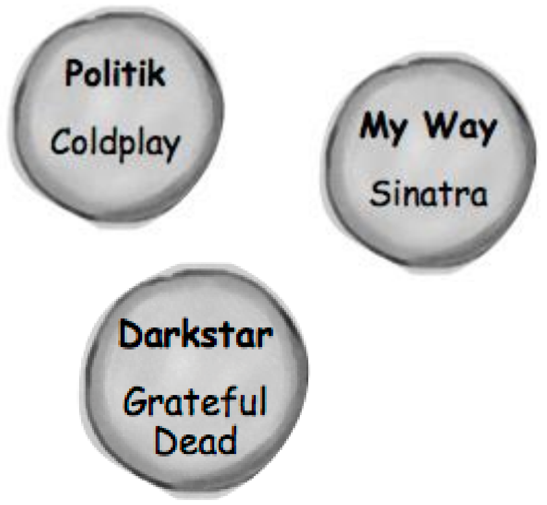
Classes and Objects

Each class that is used to create objects contains:

- *Instance variables*: descriptive characteristics or states of the object
- *Methods*: behaviors of the object, i.e., what it can do

Each object made from the class has its own (different) values for the instance variables of that class





Objects

An object has:

- *state* - descriptive characteristics (instance variables)
- *behaviors* - what it can do or what can be done to it (methods)

The state of a bank account includes its account number and its current balance

The behaviors associated with a bank account include the ability to make deposits and withdrawals

Note that the behavior of an object often changes its state

Objects and Classes

A class
(the concept)

Bank
Account

An object
(the realization)

John's Bank Account
Balance: $5,257

Bill's Bank Account
Balance: $1,245,069

Mary's Bank Account
Balance: $16,833

Multiple objects
from the same class

Quick Check

What am I? (class, object, method)

I behave like a template	class
I have a state	object
I am located in objects	method
My state can change	object
I define methods	class
I have behaviors	class, object

Quick Check

What is wrong with the following class definition?

```
public class Program1
{
    public static void main(String[ ] args)
    {
        System.out.println("My first Java program");
    }
}
```

Executable statements end with a “;”

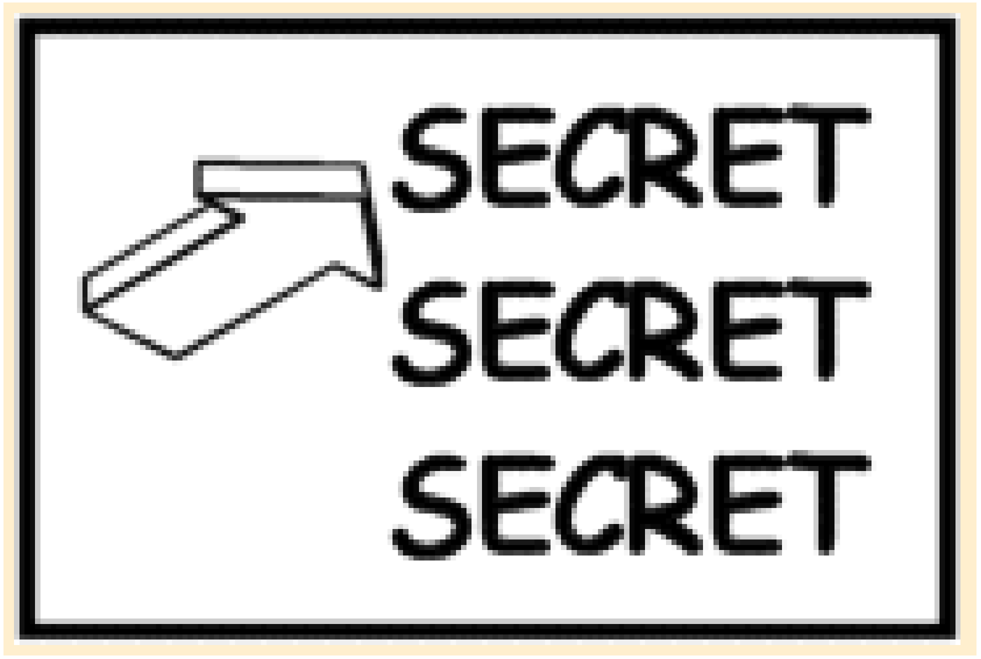
Quick Check

What is wrong with the following class definition?

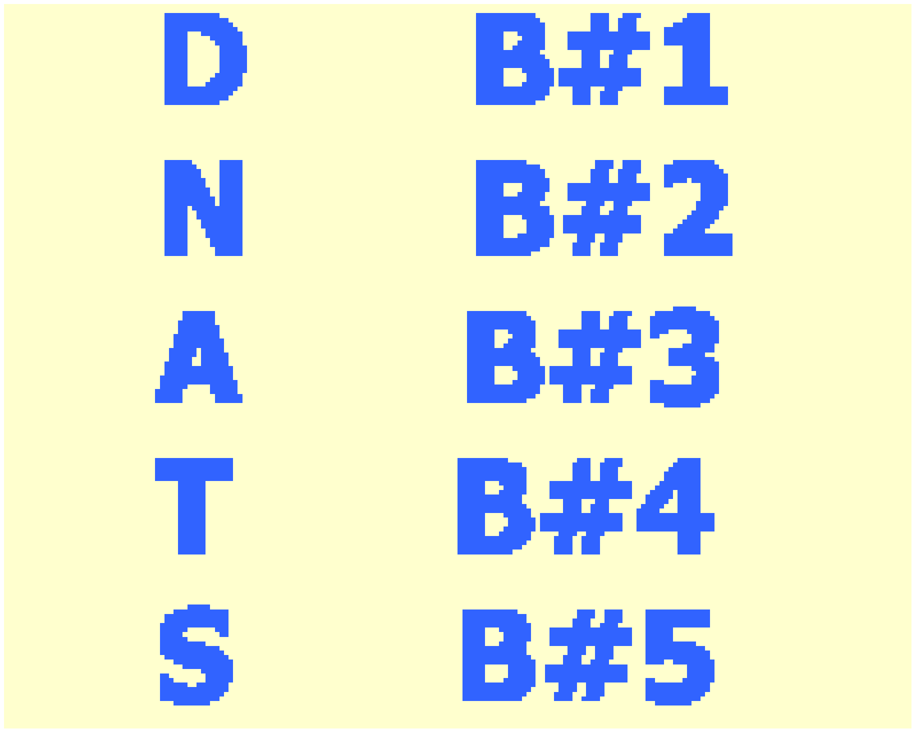
```
public class Program2
 {
    public static void main(String[] args)
    {
       System.out.println("My second Java program");
    }

  }
```

The definition of a class is placed within { }, which are missing

REBUS PUZZLE



Top Secret



D	B#1
N	B#2
A	B#3
T	B#4
S	B#5

Group Exercises

Ex: 1.3

Ex: 1.15

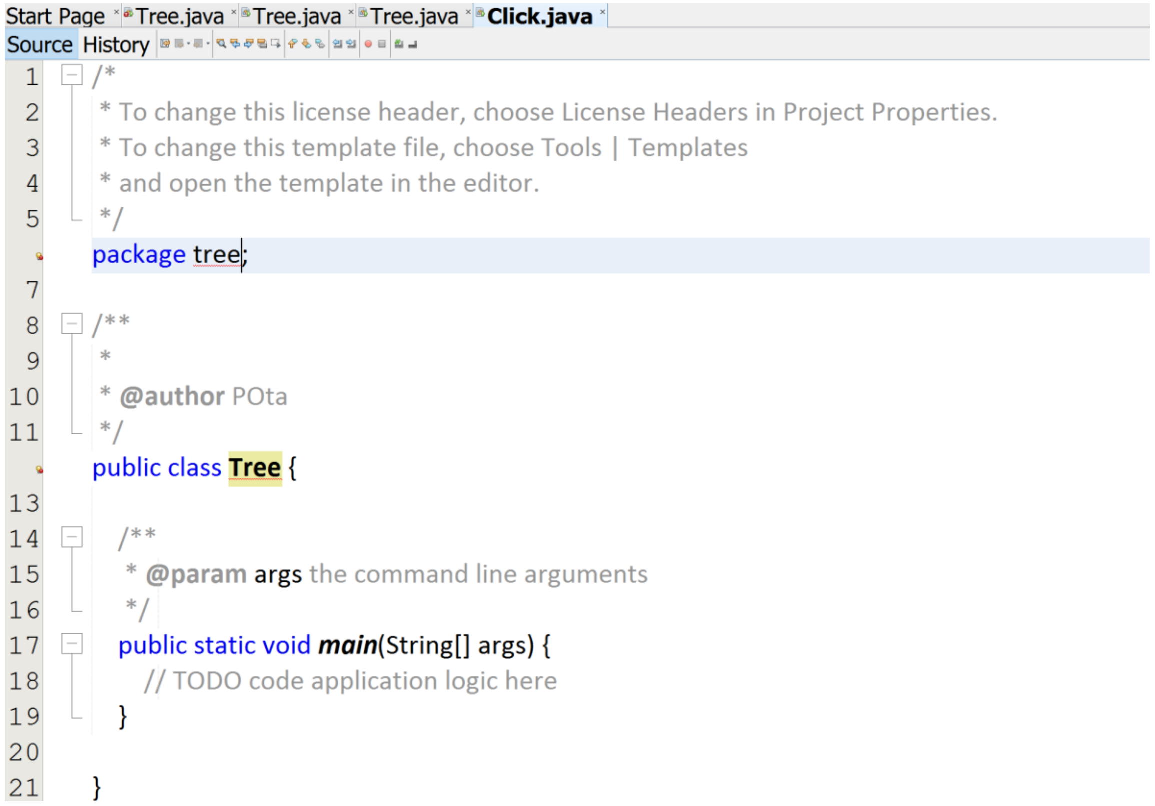
Ex: 1.16

Ex: 1.20

PP: 1.6

Getting Started with PP 1.6

1. *Start NetBeans IDE or Eclipse*
2. *Choose File > New Project*
3. *Click Next*
4. *Type Tree in Project Name box*
5. *Click Finish*

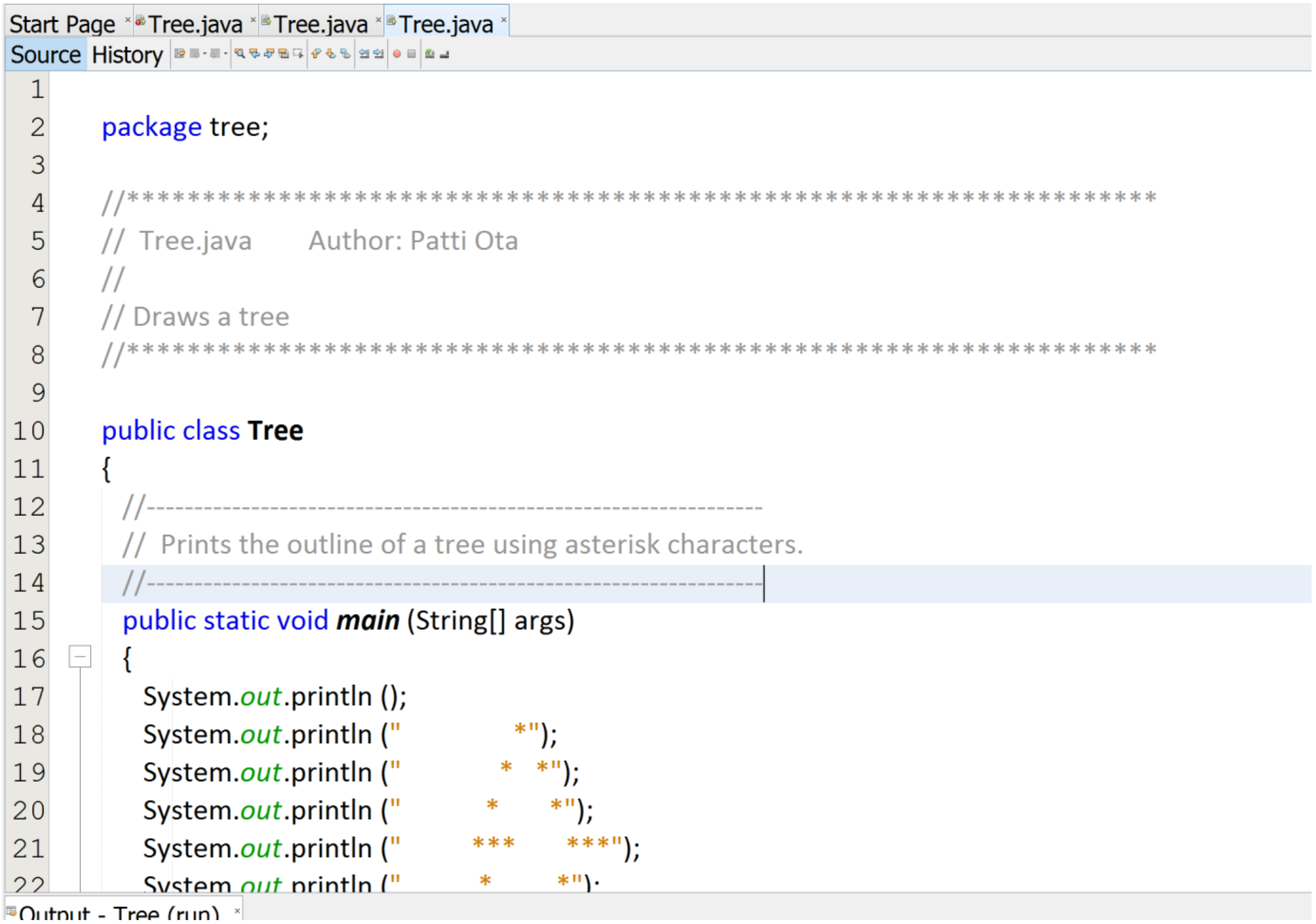


```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package tree;

/**
 *
 * @author POta
 */
public class Tree {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```

6. *Delete the code*
7. *Copy and paste the Lincoln class (on D2L)*

```
/***************************************************************************
// Lincoln.java Author: Lewis/Loftus
//
// Demonstrates the basic structure of a Java application.
//***************************************************************************

public class Lincoln
{
//------------------------------------------------------------------------
// Prints a presidential quote.
//------------------------------------------------------------------------
public static void main (String[] args)
    {
    System.out.println ("A quote by Abraham Lincoln:");

    System.out.println ("Whatever you are, be a good one.");
    }
}
```

My Example



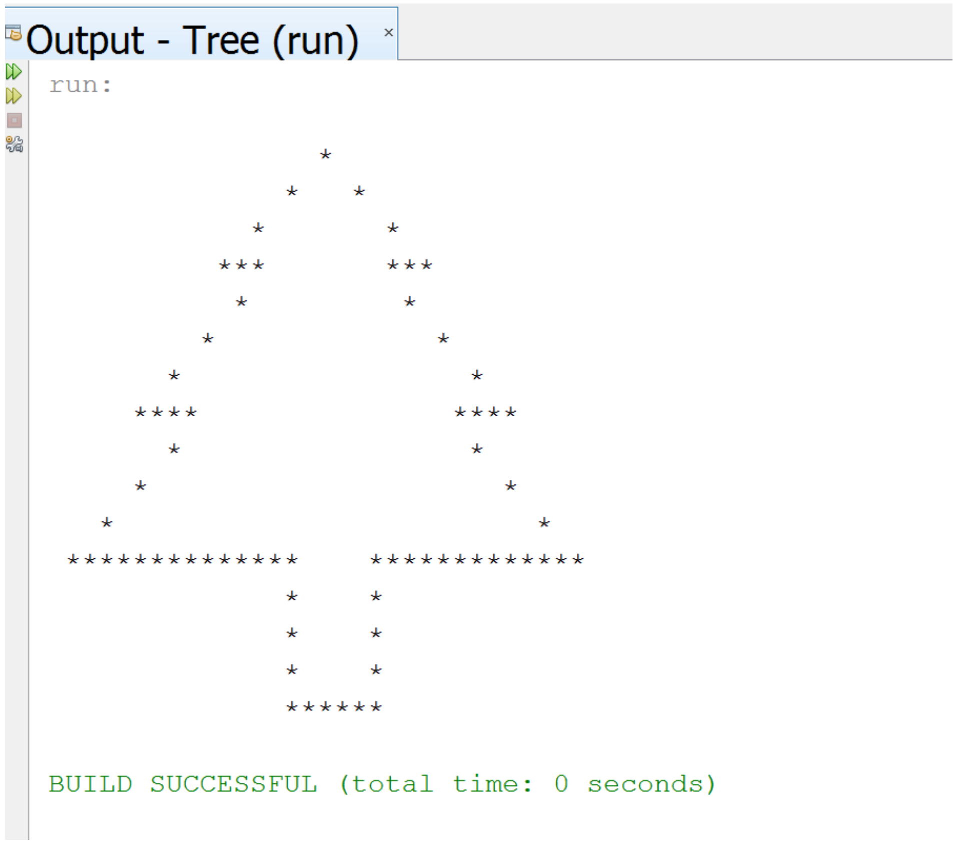
Start Page Tree.java Tree.java Tree.java

Source History

```

package tree;

//*****************************************************************
//  Tree.java       Author: Patti Ota
//
// Draws a tree
//*****************************************************************

public class Tree
{
  //-----------------------------------------------------------------
  //  Prints the outline of a tree using asterisk characters.
  //-----------------------------------------------------------------
  public static void main (String[] args)
  {
    System.out.println ();
    System.out.println ("          *");
    System.out.println ("         *  *");
    System.out.println ("        *     *");
    System.out.println ("       ***     ***");
    System.out.println ("        *      *");
```

Output - Tree (run)

Output



```
run:

                *
              *   *
            *       *
          ***       ***
           *         *
         *             *
       *                 *
     ****               ****
       *                 *
     *                     *
   *                         *
 **************    *************
              *    *
              *    *
              *    *
              ******

BUILD SUCCESSFUL (total time: 0 seconds)
```

Assignment for Class 3

Read Chapter 2.1, 2.2