Introduction

- Objectives - when we have completed this introduction to computing, you should be able to:
  - Understand the basics of software and its relationship to hardware
  - Write simple Java programs
  - Edit, compile, and run Java programs using jGRASP
  - Set a breakpoint and step through your program in debug mode
  - Use Javadoc comments in your programs
  - Run Checkstyle to verify your comments and format
  - Generate documentation for your programs

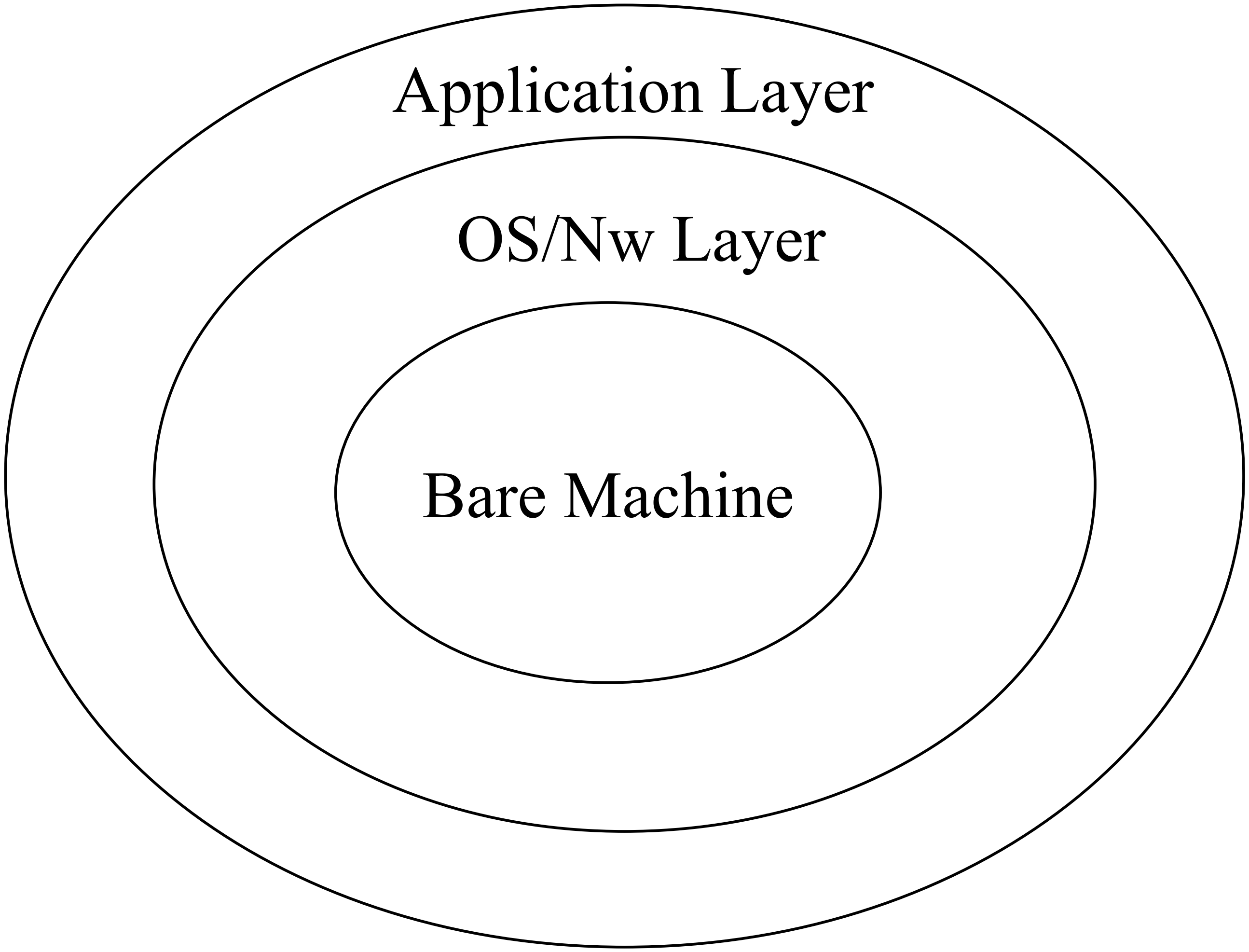
Background

- Computer System
  - Hardware and Software
- Hardware
  - “Physical” processor, memory, I/O devices, …
- Software
  - “Abstract” instructions and data stored electronically
  - Program instructions are human readable as text and machine readable as executable binary
- Computing
  - “The Act of” - Software running (executing) on hardware, processing input and producing output to solve a problem, entertain, communicate, etc.
- Fields/Disciplines of Computing
  - CS + SwE (incl WRSwE) + CpE + IS + IT + …

Sw and Hw Relationship

- Bare Machine
  - All physical components, devices, microcode
- OS/Network Layer
  - All system software: OS, Network, device drivers (Windows, Linux, MacOS, UNIX)
  - Management of all hardware: processor, memory, I/O devices
  - Management of all running software (multiple processes)
- Application Layer
  - All software applications: MS Office, Internet browsers, IDEs (Integrated Development Environments), compilers, …, including **programs written in this course**

Sw and Hw Relationship



Software

- In this course
  - Hw is assumed; designed/implemented by CpE, EE, physicists, etc.
  - Sw is our focus; designed/implemented by CS, SwE, IS, etc.
- Developing Sw is about
  - Problem solving
  - Design, construction, testing, …
  - Managing the inherent complexity
  - Organizing the algorithms (instructions) and data as classes and objects in object-oriented programming

Object-Oriented Concepts

- Classes
- Objects
- Encapsulation
- Inheritance
- Polymorphism
- Exception Handling

All of these OO concepts are directly supported in the Java programming language

Java

- A *programming language* specifies the words and symbols that we can use to write a program
  - Employs a set of rules *(syntax)* that dictate how the words and symbols can be put together to form valid *program statements*
  - Defines the meaning (semantics) of *program statements*
- Java was created by Sun Microsystems and introduced in 1995 (acquired by Oracle, 2010)
- Java continues to evolve and grow in importance to the software industry

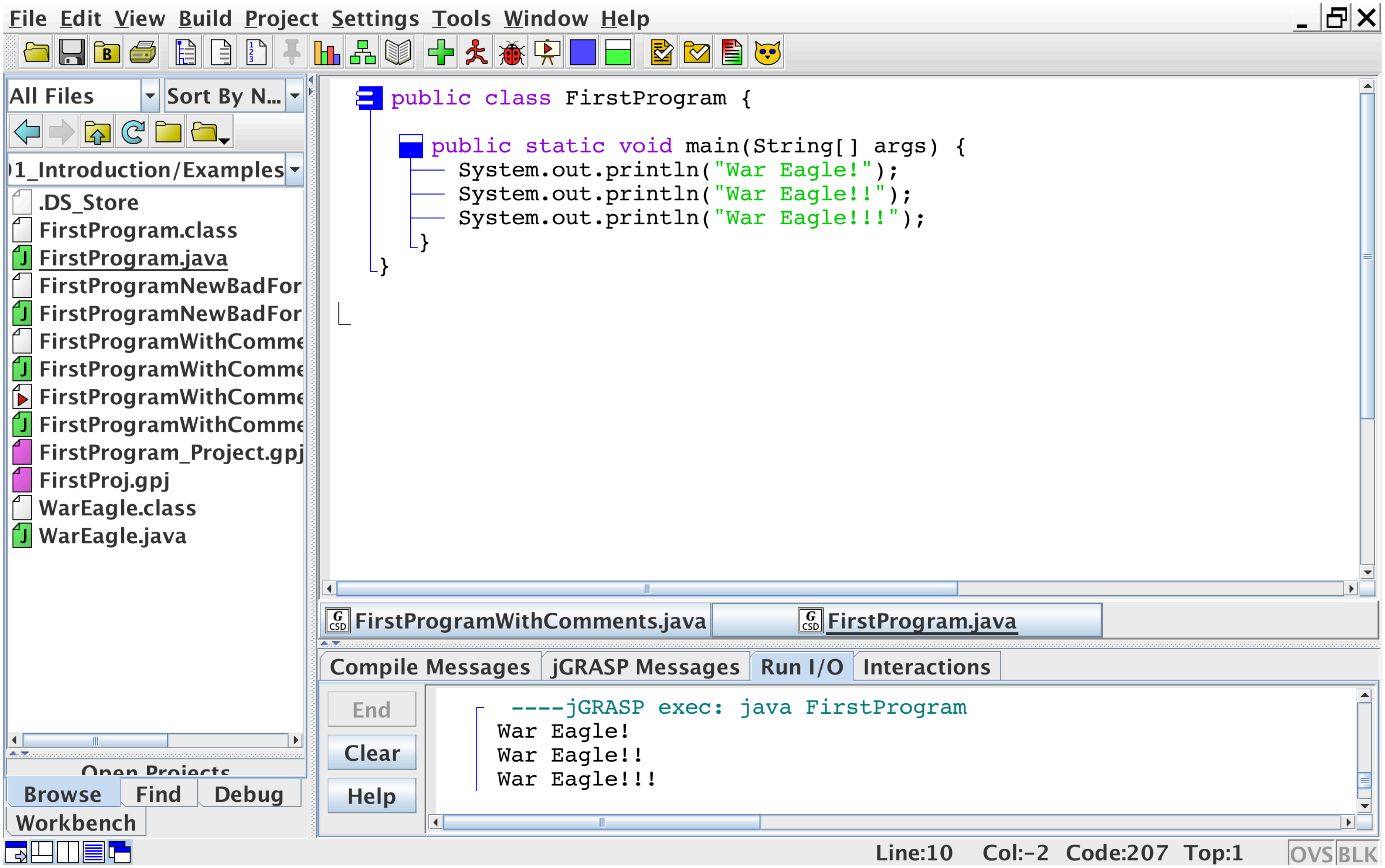
Java Program Structure

- In the Java programming language:
  - A ***program*** is made up of one or more *classes*
  - A ***class*** contains zero or more data and/or *methods*
  - A ***method*** contains zero or more local data and/or program *statements* that form an *algorithm*
- These terms will be explored in detail throughout the course
- A Java application has a class containing a method called **`main`**

First Program with jGRASP

1. Start up jGRASP
2. Open a new file
3. Enter the program (incrementally: steps 3-6)
   - The program should print “War Eagle” three times
4. Save program
5. Compile program
6. Run program (check for correct output)
7. Set a breakpoint and Debug (step through each statement
8. Generate the control structure diagram (CSD) and Documentation; turn on/off line numbers

jGRASP

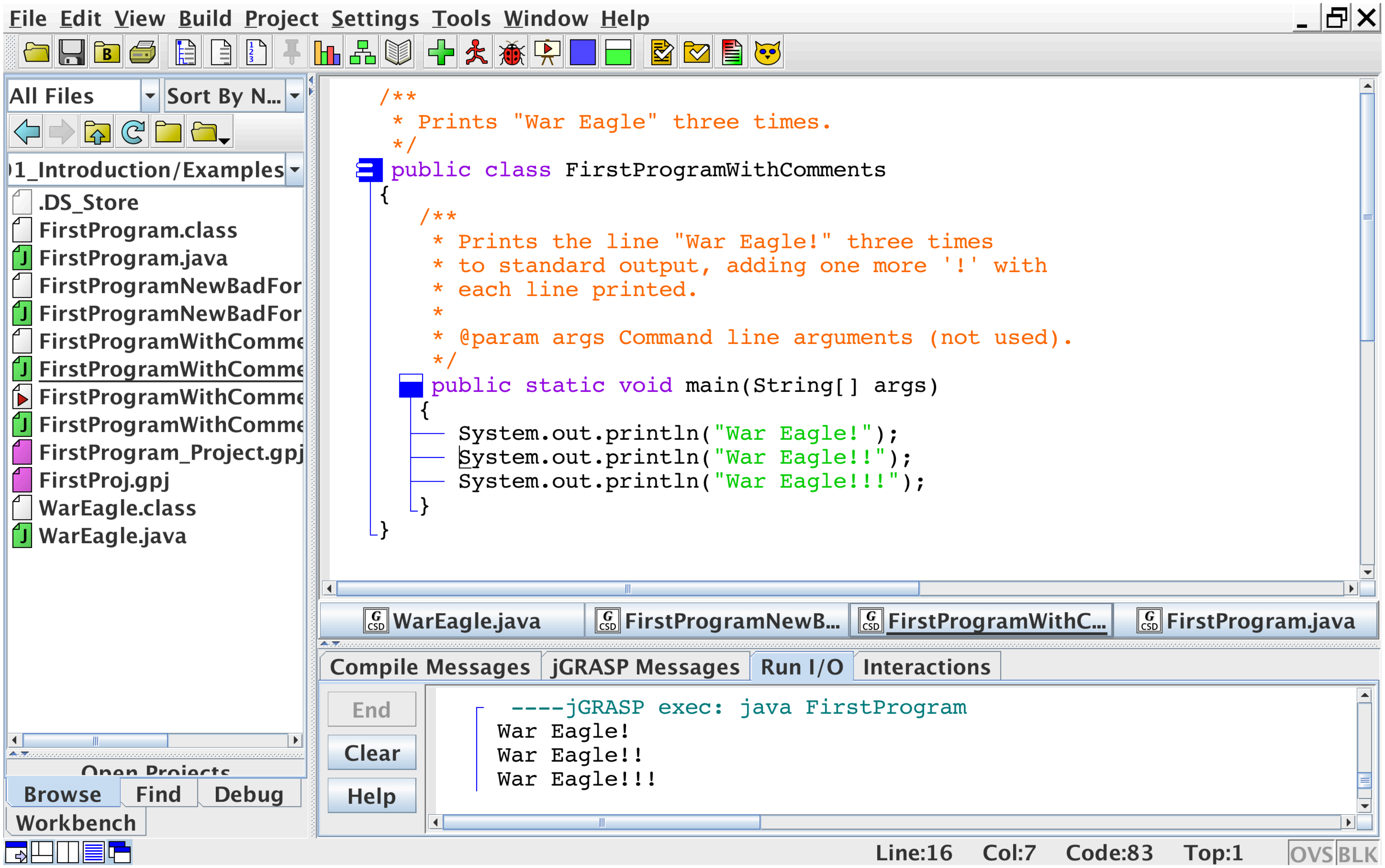


```
public class FirstProgram {

   public static void main(String[] args) {
      System.out.println("War Eagle!");
      System.out.println("War Eagle!!");
      System.out.println("War Eagle!!!");
   }
}
```

```
 ----jGRASP exec: java FirstProgram
War Eagle!
War Eagle!!
War Eagle!!!
```

FirstProgram.java

jGRASP



```
   /**
    * Prints "War Eagle" three times.
    */
 public class FirstProgramWithComments
 {
     /**
      * Prints the line "War Eagle!" three times
      * to standard output, adding one more '!' with
      * each line printed.
      *
      * @param args Command line arguments (not used).
      */
    public static void main(String[] args)
    {
       System.out.println("War Eagle!");
       System.out.println("War Eagle!!");
       System.out.println("War Eagle!!!");
    }
 }
```

```
  ----jGRASP exec: java FirstProgram
War Eagle!
War Eagle!!
War Eagle!!!
```

FirstProgramWithComments.java

Software Concepts

Algorithms and Data

Dissecting a Java Program

Program Development, Translation, and Execution

Syntax, Semantics, and Errors

Overview of Programming Languages

Object-Oriented Programming

Algorithms and Data

- Sw ::= algorithms (“instructions”) and data
- Algorithms ::= Sequence, Selection, Iteration of instructions
- Pseudo-code (initial prog. design) becomes “formal” program (i.e., code in a programming language like Java)
  - Pseudo-code can become comments in the program
- Many pieces of code for algorithms and data
- Organized into classes which define objects (Object-Oriented Programming)

Dissecting a Java Program

```
/**
 * Prints the line "War Eagle!" three times
 * to standard output.
 *
 * @author James Cross
 * @version e.g., date written
 */
public class FirstProgram
{
   /**
    * Prints "War Eagle!" three times.
    *
    * @param args Command line arguments (not used).
    */
   public static void main(String[] args)
   {
      System.out.println("War Eagle!");
      System.out.println("War Eagle!!");
      System.out.println("War Eagle!!!");
   }
}
```

Parts of this Program

- Comments
- Class
- **`main`** Method
- Identifiers
  - Reserved Words
  - Other (e.g., method and variable names)
- Java API
- Literals
- White space

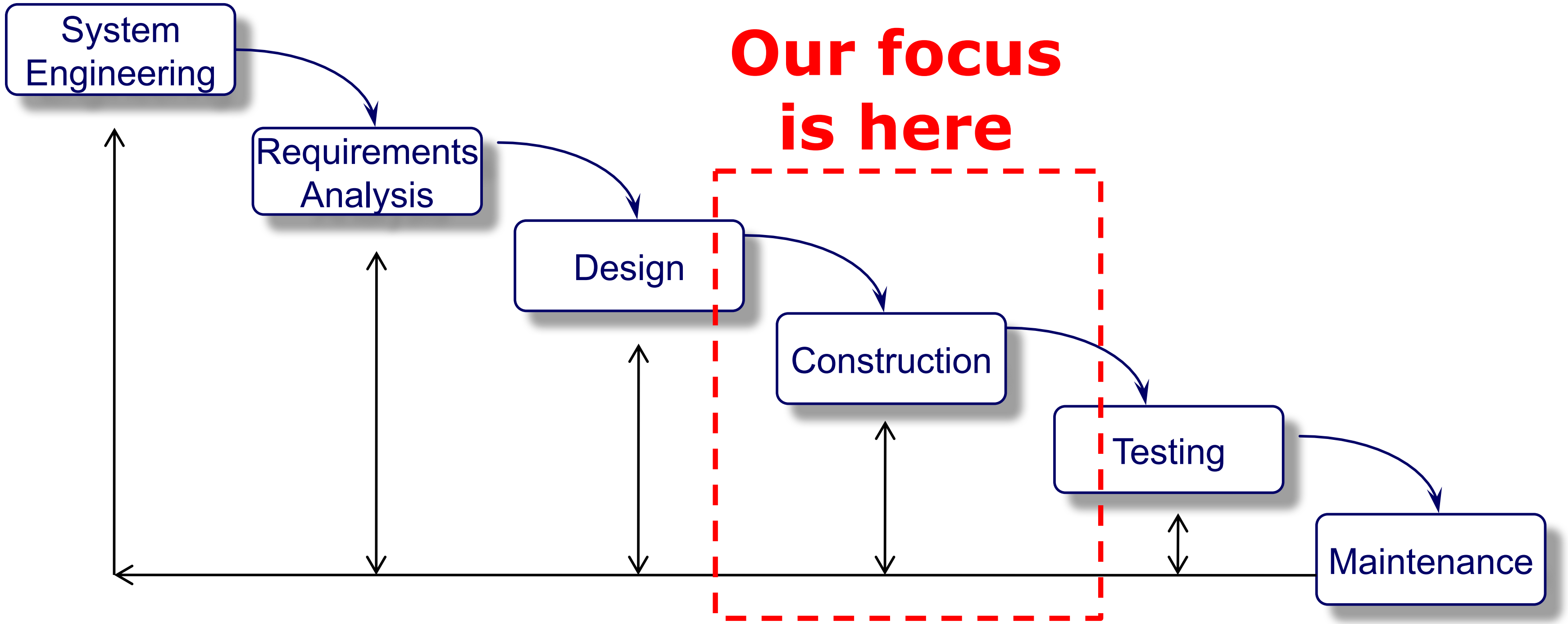
Identifiers can be any combination of letters, digits, dollar sign ($) and underscore (_) characters; cannot begin with a digit. Java is “case sensitive”.

```
/**
 * Prints the line "War Eagle!" three times
 * to standard output.
 *
 * @author James Cross
 * @version e.g., date written
 */
public class FirstProgram
{
   /**
    * Prints "War Eagle!" three times.
    *
    * @param args Command line arguments (not used).
    */
   public static void main(String[] args)
   {
      System.out.println("War Eagle!");
      System.out.println("War Eagle!!");
      System.out.println("War Eagle!!!");
   }
}
```

Q1 Q2 Q3

Program Development

- There's more to developing software than coding (a.k.a. construction or implementation)



- Many variants of the process model

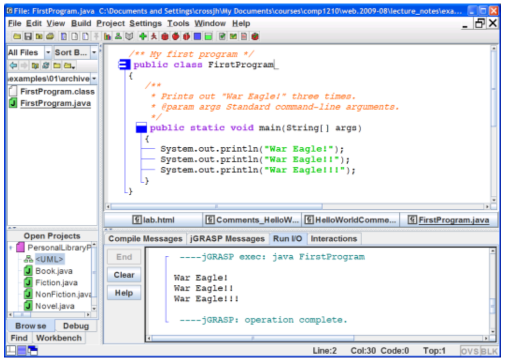
Program Development (cont.)

Construction – includes Code and Unit Test

- Code
  - Writing source code that will be compiled into an executable program.
  - **Coding standard**: Rules as to how source code should be formatted and documented - makes code easier to read and debug.
- Test (Unit Test)
  - Once you write your program, make sure that the **actual output** of your program matches the **expected output** as specified in the requirements document.

Program Development (cont.)

- Program development tools are valuable aids during the process.
  - A good IDE (integrated development environment) with program editor, debugger, interactions, etc.
    will should become one of your best sw tools.
  - *jGRASP* (jgrasp.org) with Java, Checkstyle, JUnit, Web-CAT



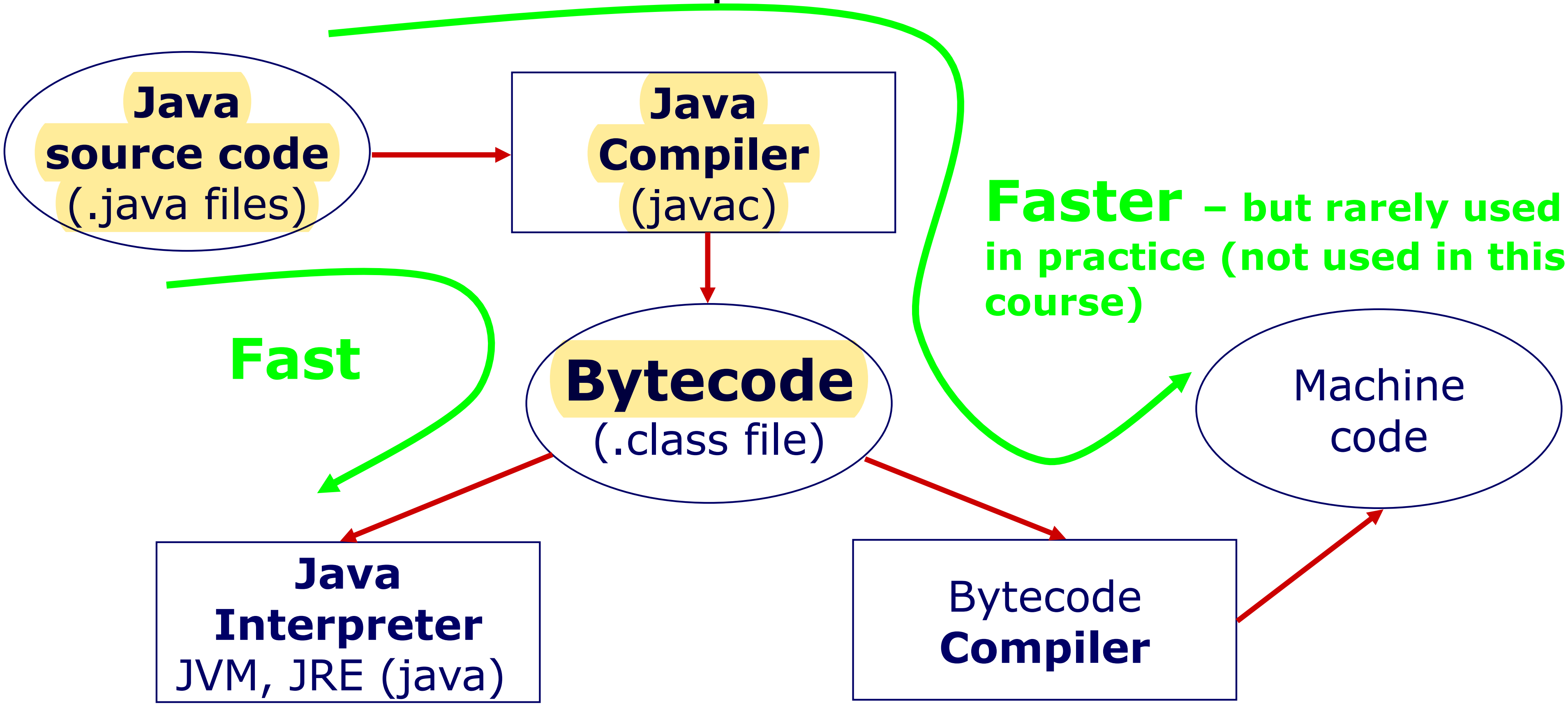
```java
/** My first program */
public class FirstProgram
{
   /**
    * Prints out "War Eagle!" three times.
    * @param args Standard command-line arguments.
    */
   public static void main(String[] args)
   {
      System.out.println("War Eagle!");
      System.out.println("War Eagle!!");
      System.out.println("War Eagle!!!");
   }
}
```

```
 ----jGRASP exec: java FirstProgram

War Eagle!
War Eagle!!
War Eagle!!!

 ----jGRASP: operation complete.
```

  - *Checkstyle* is used with jGRASP to support the coding standard we'll use in this course.

Program Translation

- **Compiler** v. **Interpreter (Java Virtual Machine)**
- The Java translation process:



Java source code (.java files)

Java Compiler (javac)

Bytecode (.class file)

Java Interpreter JVM, JRE (java)

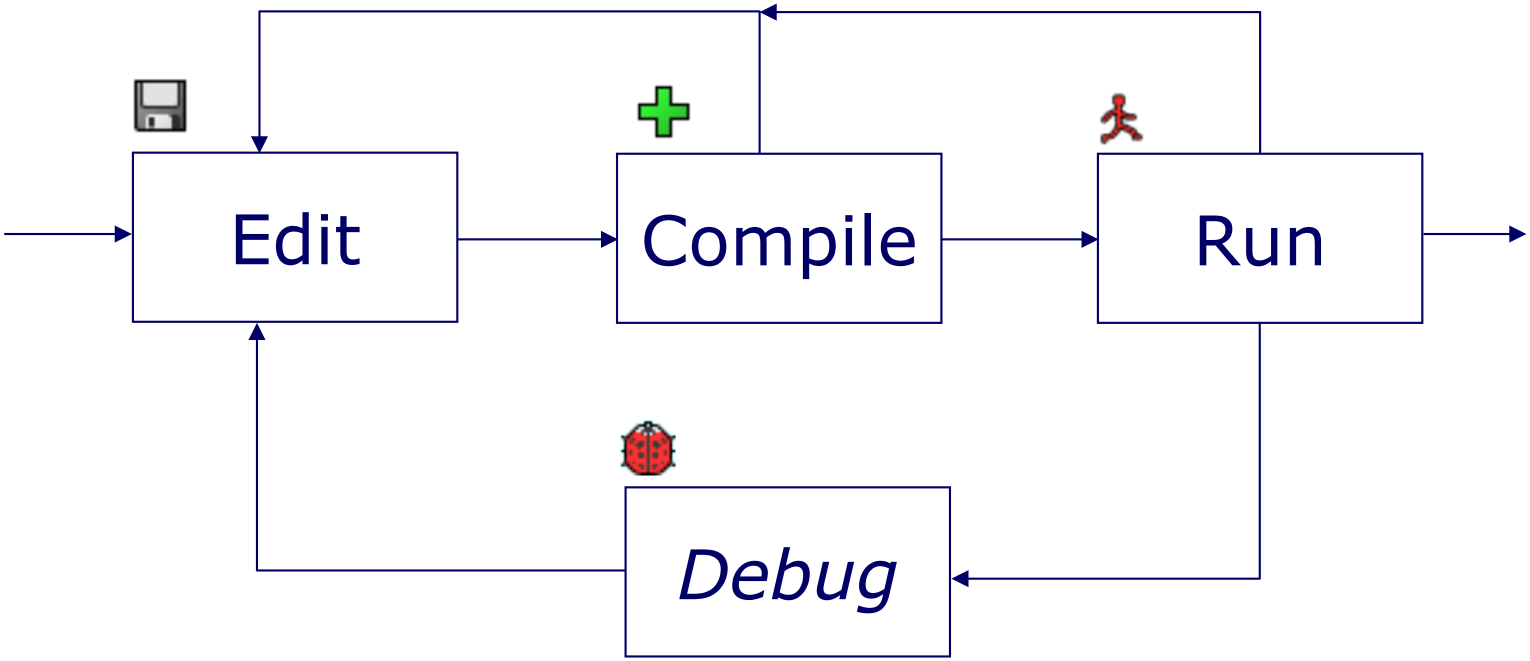
Bytecode **Compiler**

Machine code

Fast

Faster – but rarely used in practice (not used in this course)

The Implementation Cycle...



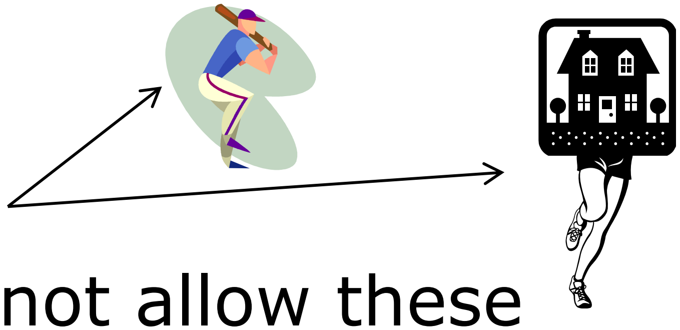
- This cycle implies incremental program construction.
- Plan to repeat this cycle early and often.

Syntax and Semantics

- Syntax: “grammar”
  - Rules of how the vocabulary can be used to compose legal structures in the language.
  - In the context of programs, the language syntax describes how to form legal statements and other constructs in the language.
- Semantics: “meaning”
  - What a given legal structure in the language means.
  - In the context of programs, the language semantics describes what will happen when a legal statement in the language is executed.

Syntax and Semantics (cont.)

- In natural languages, some things can be syntactically correct but have no meaning…
  - Blue ideas sleep furiously.
- … or be syntactically correct but have many (possible) meanings.
  - Time flies like an arrow.
  - The house flies like a saucer.
  - Did you ever see a home run?



- Programming languages do not allow these situations - - there is no ambiguity!
  - A program will have the same behavior each time it is run - - assuming input, if any, is the same.

Program Errors Q4 Q5 Q6

- **Compile-time errors**
  - Compilation cannot be completed
    - Syntax errors
    - Static semantic errors
  - The Java compiler will not produce bytecode.
- **Logical errors (logic errors)**
  - Execution proceeds and halts normally, but incorrect behavior or incorrect results are observed.
- **Run-time errors**
  - Execution is halted abnormally.
    - Deep-end, crash, blow up, crash and burn, hosed
  - Illegal operations, exceptions.
- Find errors by **testing** and remove them by **debugging**

Overview of Programming Languages

- A programming language is an artificial language designed for humans to express programs and have these programs translated into machine-executable form.
- Programming languages can be categorized in different ways, for example:
  - Machine languages
  - Assembly languages
  - **High-level languages (e.g., Java, C++, Python)**
- Languages in different categories are obviously going to be very different from each other, but even languages within the same category can vary widely.

Same Program, Different Languages

Java

```
/** Prints a quote from the Plains */
public class War_Eagle
{
   public static void main(String[] args)
   {
      System.out.println ("War Eagle!\n");
   }

}
```

C

```
/* Prints a quote from the Plains */
main()
 {
    printf ("War Eagle!\n");
 }
```

Ada

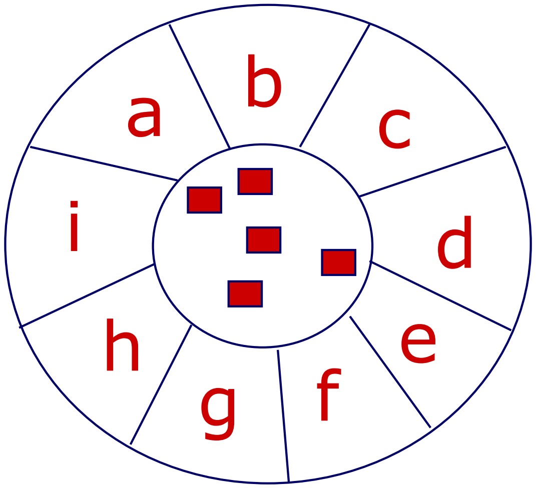
```
-- Prints a quote from the Plains
with Ada.Text_IO;
use  Ada.Text_IO;
procedure War_Eagle is
begin
   Put ("War Eagle!");
   New_Line;
end War_Eagle;
```

Perl

```
# Prints a quote from the Plains
print "War Eagle!”, “\n";
```

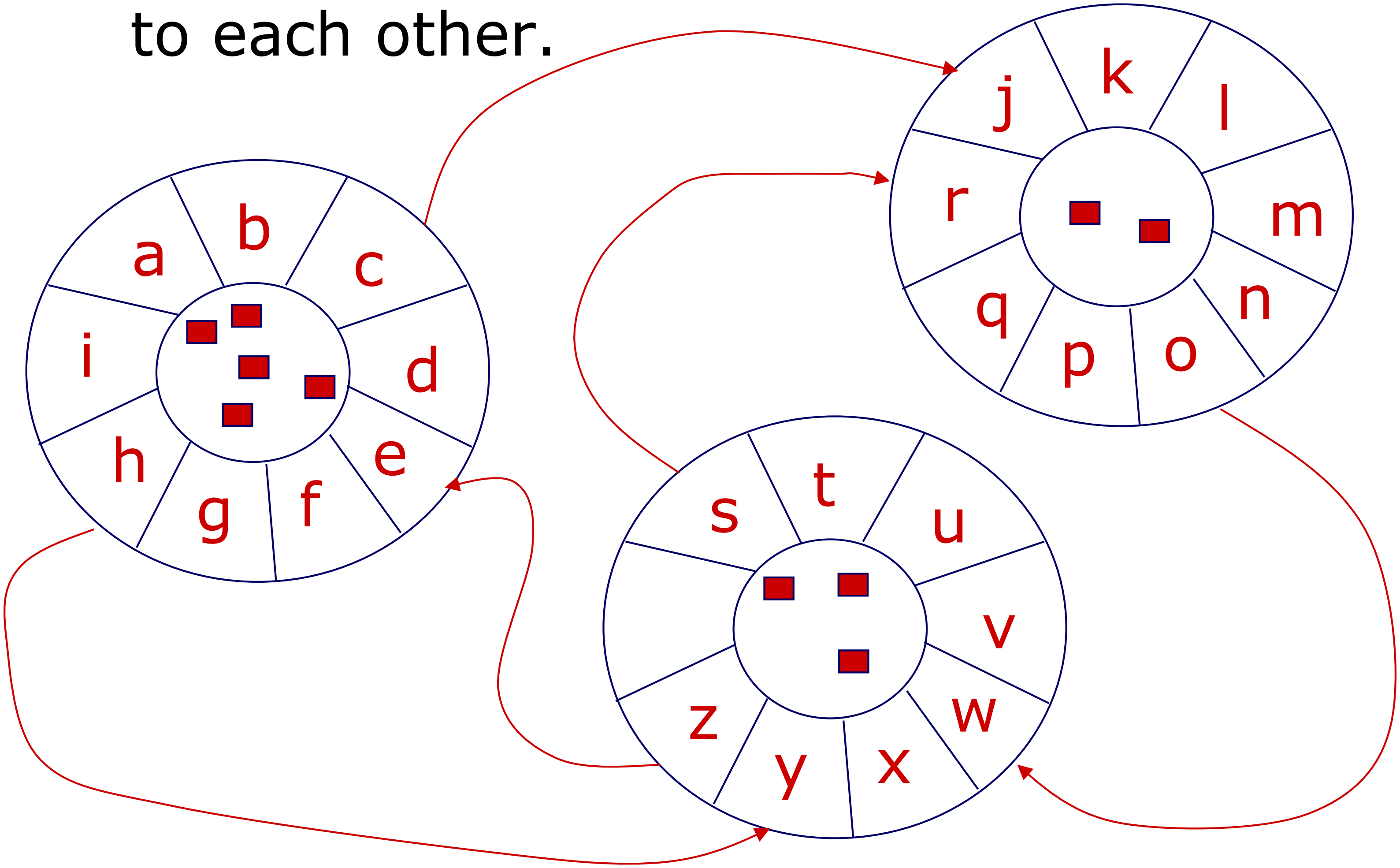
Object-Oriented Programming

- OOP is a programming world-view in which things in the real world are modeled as software **objects**.
    - An object is really just an **abstraction** of a real-world thing, implemented as an **encapsulation** of private **data** and **methods** (operations on that data).



Object-Oriented Programming (cont.)

- Objects communicate by sending **messages** to each other.



Object-Oriented Programming (cont.)

- **Class** = a description of an entire category or group of objects
  - Classes model categories of real world things by describing their "data" and their "operations."

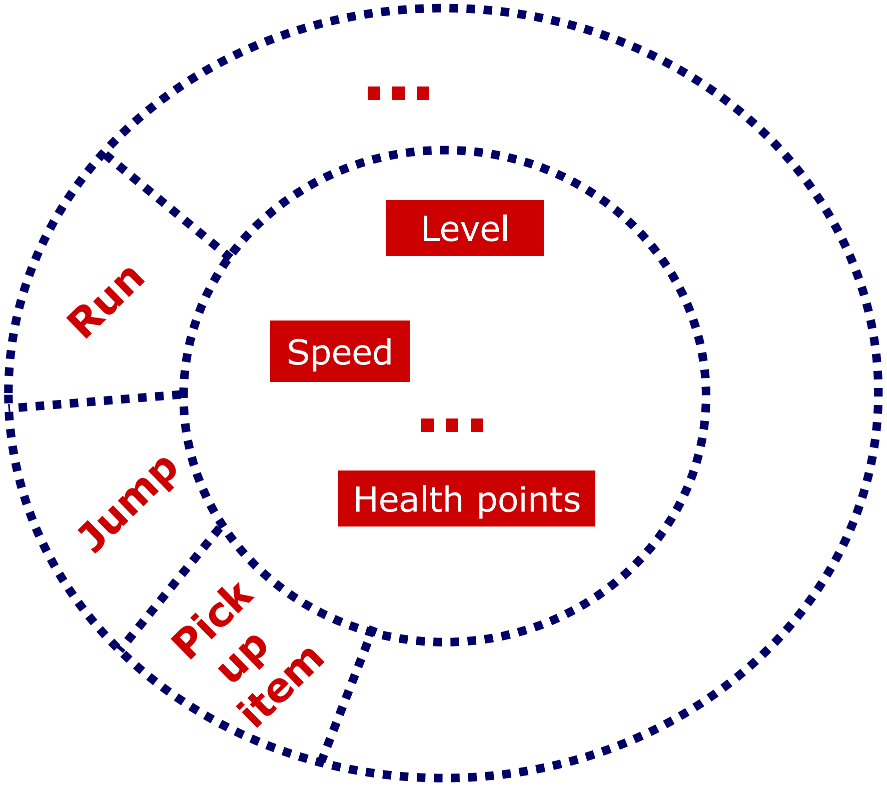
Class Name: GamePlayer

Data:

    Level
    Speed
    Health points
    …

Operations:

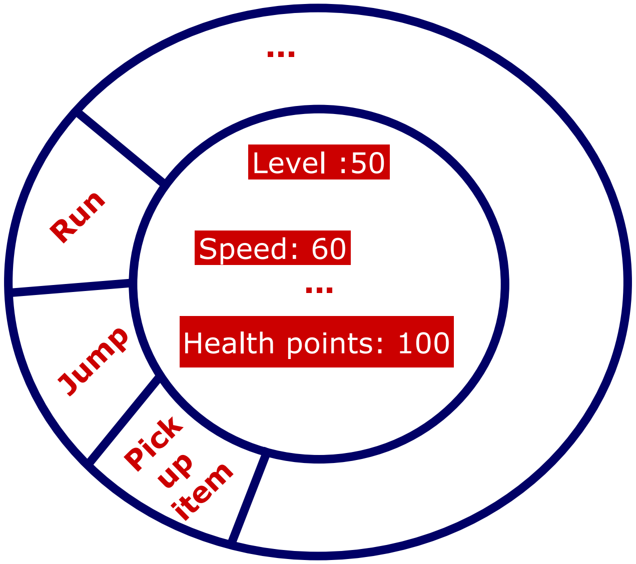
    Run
    Jump
    Pick up item
    …



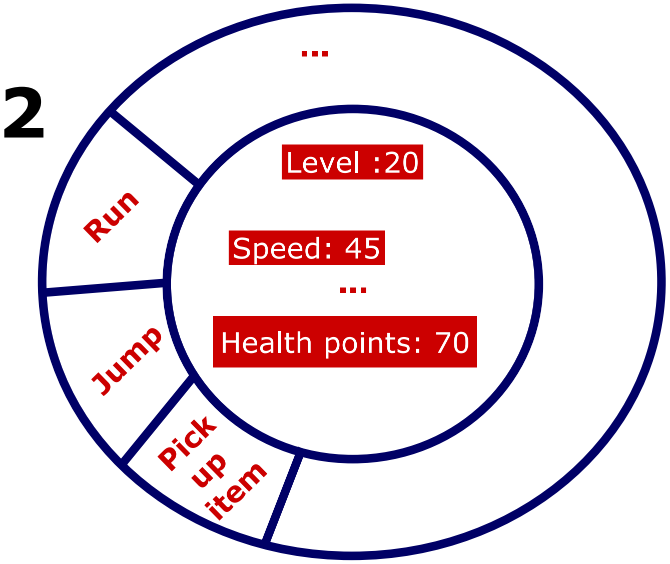
Object-Oriented Programming (cont.)

- An object is an **instance** of some particular class.

Player1



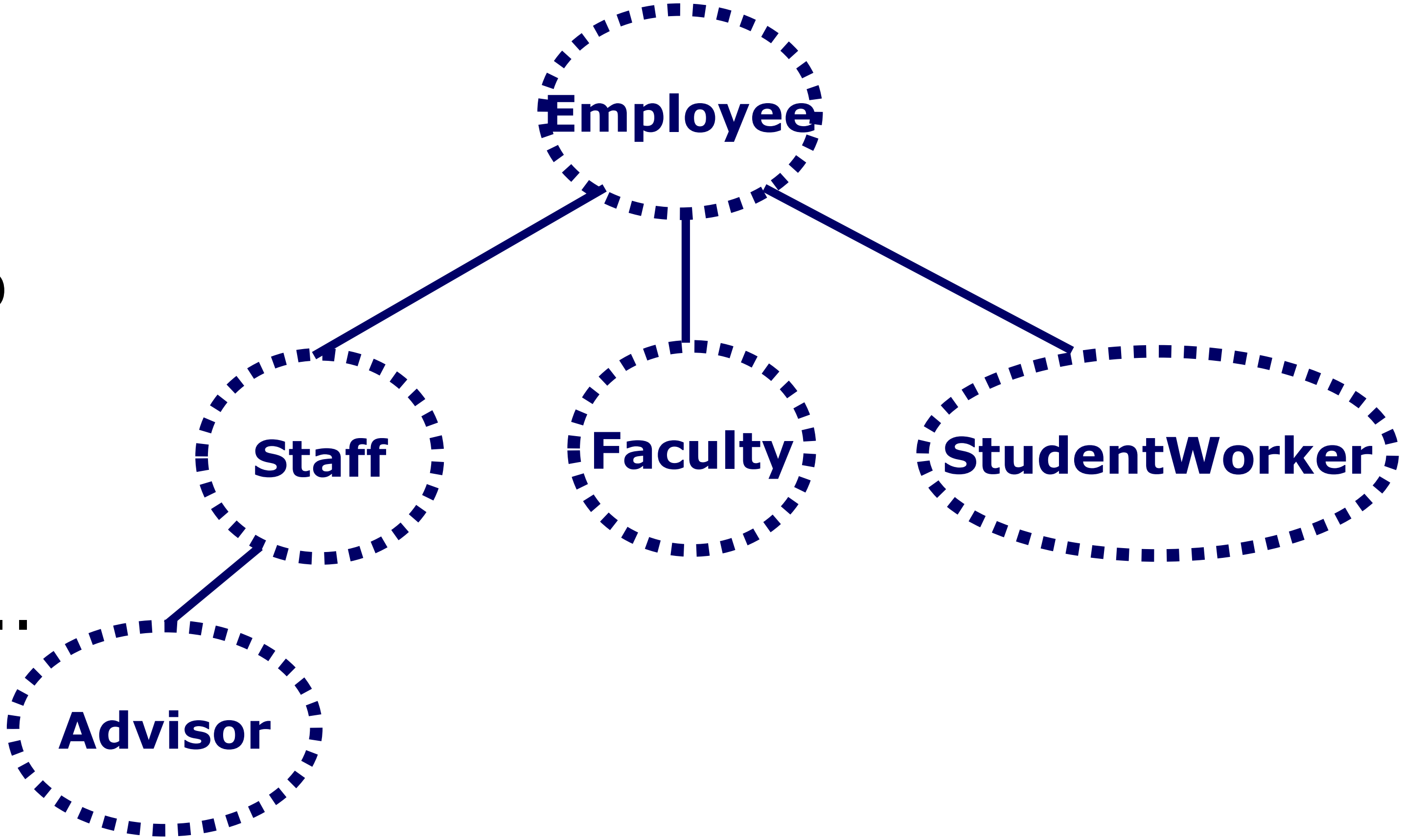
Player2



Object-Oriented Programming (cont.)

- New classes are derived from existing ones through **inheritance**.

Suppose you were creating a program to track employees at Auburn University...



Object-Oriented Programming (cont.)

- OOP is intended to support software **reuse**.
- **Class libraries** are an important element of this support.
  - Class libraries are sets of classes designed to be reusable components whose services can be used by many programs.
- The Java Application Programming Interface (**API**) is a set of class libraries that comes with the **JDK**.
  - The Java API is organized into **packages** such as **`java.awt`**, **`java.io`**, **`java.lang`**, and **`java.net`**
  - Example: The System class that you use in your output statements is in the java.lang package