**NOTES**:

*import.packageName.\* // importing all classes within a package*

java.util.\*

java.swing.\*

java.awt.event.\*

java.lang.\* (does not need to be imported, automatically)

String Class

*String != array of characters // cannot manipulate characters within string*

*String name = new String(“stuff”) // create new string*

* *String name = (“stuff”) // same*
* *String name = newString(“Barney”) // same*
* *name.concat = name+ // same*

String Comparison

String s1 = …

String s2 = ...

.equals

* if(s1.equals(s2)) *// returns true*

Standard I/O (Command Line)

Output:

* System.out.println(string) *// adds new line*
* System.out.print(string) *// next thing prints on same line*

*// good for prompts*

Scanner class *// in java.util*

Different methods depending on type of data to read in

s.next Type() *// general syntax*

Int x = s.nextInt()

double y = s.nextDouble()

String name = s.nextLine()

Scanner s = new Scanner(System.in);

Static Methods

Can be invoked on no object

Classname.methodname(...) *// Syntax*

* Double y = math.sqrt(x); *// Invoked*

System class has static methods

* **System**.exit(); *// static method that invokes OS*
* System.**out**.println(...); *// Standard output*

Can declare a method to be static

Public static.newtyle name(...); *// syntax*

* Public static void **main**(String[ ], args) {}

*// first thing run when ap is involved*

Creating own Java classes

* Applications *// main method, meant to be run*

*// visual app*

* Support class
  + Logic classes *// does not run independently*

*// constricted// used by other classes*

* Decomposition *// goal is to simplify*

Apps with multiple simple classes

* + - Easier to design
    - Easier to test
    - Easier to reuse in multiple apps

Eg. **Clock** class

* Keeps track of:
  + hour/minute *// data stored*
* Allows user to: *// how to manipulate data*
  + Set hour/minute (to legal values)
  + Increment current minute (and possibly current hour)
  + Get current hour/minute

(Universal Modeling Language) UML *// visual tools for designing large programs*

* Universally agreed structure *// so anyone can understand it*
* **Class Diagram** *// commonly used is oop design*
  + Relationship between classes
  + Composition *// one class contains an instance of another class*

Member variables // internal representation

* Declared inside class but outside any method

// represent current state of object

*// exist for lifetime of object*

* Should be private *// can only be accessed within the class*

Constructor

* Set initial state of object
* Set initial values of member variable
  + Construct contained objects
  + Reading from file
  + Has SAME NAME as class
  + No return type

IMPORTANT: member vs local variable

Member = declared inside class

Local = declared inside method

Overload // giving multiple definitions to something

* Constructor - must be same as class name
* Legal iff compiler can distinguish based on

type/ number of parameters // make sure the user can as well

OBJECT ORIENTED DESIGN

* What are my main objects?
* What are their relationships?
* What are its member variables? // often have man choices. May need clarification
  + What information does it keep track of
  + How is it represented
* What constructors are needed?
  + How will state be initialized?
  + What does user need to specify?
* What methods are needed?
  + How will users manipulate?
  + Object state?

Analysis - What should I develop?

Design - How should this be implemented?

* Modifier method - change object state
* Inspector method- returns info about state

Validation - meant to avoid putting object in an illegal state as defined by specification

Inspector - returns information about object state

Debuggers (toString) - returns all member variables as a single string

JavaDoc:

* Inline documentation // API in HTML
* Comments
  + /\*\* ... \*/ //converted into API
  + /\*\* @ propertyname \*/