1. Characteristics of Java compared to C
   1. Object-oriented, not structured programming
   2. Main file: only one in the source packages
      1. public: everyone can access it
      2. static: constant
         1. In all upper-case
      3. void: no return type
   3. Same primitive data types
      1. String types instead of char arrays
   4. Has its own virtual environment
      1. Uses databases instead of text files
      2. Import statements similar to include statements from C
         1. Grab library of pre-defined stuff (i.e., classes)
         2. import java.awt.Color
   5. Bad idea to make global variables
2. NetBeans
   1. Encapsulate fields via right-click menu
      1. Getter and setter
      2. Autogenerates code
         1. Right-click +…
            1. Insert code
            2. Fix imports
            3. etc.
      3. Example
         1. private int variable;
         2. Select variable and right-click
         3. Refactor
         4. Encapsulate field
         5. Setter/Getter
   2. Red squiggles under wrong things
   3. Comment-out: CTRL + FWD slash
   4. Submitting assignments
      1. Right-click project
      2. Compress to zip
   5. Clicking on a class name will bring user to class definition.
   6. Typing “.” shows all states and behaviors.
   7. **Package**: namespace for organizing classes and interfaces in a logical manner, making them easier to manage.
      1. Can import different parts, or an entire package.
      2. **Application Programming Interface (API)**
         1. Java’s has many packages
      3. File system view directories
         1. build (.java 🡪 .class [similar to .o]): compiled Java source code files; Class files used during runtime.
         2. dist (.jar instead of .exe): executable compressed version of class files for distribution.
            1. Command: java –jar BattleshipGame.jar
            2. Can also double-click on newer versions of Java.
         3. images: images associated with project.
         4. nbproject: NetBeans-specific files.
            1. Setup files.
            2. Do not modify.
         5. src (.java): source code files.
         6. test: automatically tests source code.
            1. **J-unit test**: automatic testing of source code.
   8. Inserting files
      1. Source packages 🡪 right click 🡪 new package 🡪 name it,
      2. From hard drive, put downloaded files into .src data.
      3. Right click in folder under source packages and paste.
3. Java
   1. package: includes all elements, with differing functionality.
      1. Always leave main in its own package
      2. core: supporting back-end behavior
      3. user interface (UI)
   2. Naming conventions
      1. class: first name starts with an upper-case letter
      2. constructors: same name as class.
4. Key Terminology
   1. **Object**: software bundle of related state and behavior.
      1. Used to model real-life objects
      2. State and behavior are intimately related
         1. **State**: characteristics of the object.
            1. Dog

Has a tail?

Height

Weight

* + - * 1. **Instance variables** **(member variables/fields)**: form of attributes specific to they object they represent.
      1. **Behavior**: actions of the object.
         1. Dog

Barks

Runs

* + - * 1. **Methods (functions)**: task performance specific to a task.

Update the state of an instance variable.

Perform data analysis.

**Method call**: calling a method to perform tasks defined in the method.

* + 1. Why objects?
       1. **Modularity**: source code for an object can be written and maintained independently of source code for other objects.
          1. Once created, object can be easily passed around inside system.
          2. core can be changed in background while keeping same UI.
       2. **Information-hiding**: by interacting only with an object’s methods, the details of its internal implementation remain hidden from the outside world.
          1. **Encapsulation**: classes and the objects created from them encase their attributes and methods (state and behavior).

Objects can communicate with one another publicly.

How objects are made is private.

**Access-level modifiers**: public, private, an default.

**public**: everyone can access it.

class must be public.

**private**: only the class itself can access it.

Instance variables within class.

By default if access-level modifier is not included.

* + - 1. **Code re-use**: if an object already exists, you can use that object in your program.
         1. Implement/test/debug complex, task-specific objects
         2. Trust to run in your own code
         3. Pull into API easily from libraries
      2. **Pluggability and debugging ease**: if a particular object turns out to be problematic, remove it from your application and plug in a different object in its replacement instead of starting all over.
  1. **Class**: blueprint from which individual objects are created.
     1. Models state and behavior of real-world object.
     2. In C
        1. Essentially a struct with multiple variables.
        2. Related functions within the struct.
     3. In Java
        1. public class Dog {
           1. //state using instance variables
           2. String name;
           3. boolean hungry;
           4. //behaviors using methods that can be called
           5. public Boolean isHungry() {

return hungry;

* + - * 1. }
        2. public void setName (String name) {

this.name = name;

* + - * 1. }
      1. }
    1. **Instantiation**: creating an object of the defined class.
       1. **Instances**: created object of a class, possibly with differing instance variables.
       2. **Constructors**: create and initialize the object by allocating memory.
          1. Characteristics

No return type

Same name as class

If used outside of class, made public.

* + - * 1. Default no argument constructor: only allocates memory.

Included if you do not write a constructor explicitly.

* + - * 1. Writing a constructor explicitly

Allows writing instructions for initialization, etc.

* + - * 1. Writing a constructor explicitly and keeping default no argument constructor

Must write default constructor explicitly

Used to insert specified and default data

* + - * 1. Calling a constructor

Requires use of the keyword new.

* + - 1. In Java
         1. Class instance new Class();
         2. Dog puppy = new Dog();

Dog: defined class

new: keyword to create object

Dog(): no-argument constructor.

* 1. **Inheritance**: creating a new class (**subclass**) using an existing class (**superclass**).
     1. Powerful mechanism for organizing and structuring software.
        1. Create a subclass from superclass
           1. Certain amount of states and behavior in common between different subclasses derived from the same superclass.
        2. Add to it
           1. Differing features between subclasses derived from the same superclass that make them unique.
        3. Code for subclasses is easy to read.
        4. Document the state and behavior that each superclass defines, since that code will not appear in the source code of each subclass.
     2. Use extends keyword.
        1. Can only have 1 DIRECT superclass after extends keyword.
           1. Each superclass has the potential for an unlimited number of subclasses if the process is continued.
        2. Do not need to write extends Object because it is automatic.
           1. equals(Object o)

Checks if classes are the same.

* + - * 1. hashCode()

Unique identifier for each object

* + - * 1. notify()/notifyAll()/wait()/waitFor()

Listeners in case something happens that we are waiting for.

* + - * 1. toString()

Print something out to string

* + 1. In Java
       1. public class Subclassdog extends Dog {
          1. /\*inherited from superclass Dog
          2. …
          3. \*/
          4. //extras unique to Subclassdog
       2. }
  1. **Interface**: collection of related methods with empty bodies that tell objects what to do, but NOT how to do it.
     1. Contract between a class and the outside world.
        1. Name of methods
        2. Data types for inputs and outputs
     2. Uses implements keyword in the class declaration to promise to provide behavior published by the interface.
        1. All methods provided by interface must appear in source code before class will successfully compile.
        2. A class can implement ≥0 interfaces.
           1. An interface is not always required
     3. Cannot create an instance of interface.
        1. If subclass extends superclass, subclass is an instance of superclass and can use its interface.
     4. In C: akin to function prototypes.
     5. In Java
        1. public interface IClass {
           1. public void run();
           2. public void fetch();
           3. …
        2. }
        3. //Call
        4. public class Class implements IClass {
           1. @Override
           2. public void run() {}
           3. @Override
           4. public void fetch() {}
           5. …
        5. }
        6. Right click + insert code 🡪 Generate