Head First - Java

Chapter - 1

- Statements end in semicolon: ;
- Code blocks are defined by a pair of curly braces: { }
- To declare a variable, declare the type and the name.
  For example - int age = 20
  String name = Lecan.
- The assignment operator is one equal sign. (=)
- The equals operator is two equals sign. (==)
- A While loop is the same as python, which will keep running as long as the conditions are true. Only difference is the function is defined by curly braces.

Example - While (x > 12) { ← curly braces
    x = x - 1 ; ← condition
}

- Classes and methods must be defined within a pair of curly braces.
- Java has three looping constructs - while, for, do-while. ← Not available in python
- The key to a loop is the conditional test. In Java, a conditional test is an expression that results in a boolean value.
- Everything in Java has to be in a class. Java is an Object-Oriented language like Python, meaning everything in Java is an object.
- Put a boolean test inside parentheses.
  while (x == 4) { } ← code

In Java, an if test is basically the same as the boolean test in a while loop - instead of saying, "while there's still beer...", you'll say "if there's still beer...".

System.out.print and system.out.println are two ways to print things.

The difference is that system.out.println print in a newline.

To add control structures to our statements - like - "if" - use (__) to add the condition after if.

To declare a string array, use the syntax - String[] pets = {"x", "y", "z"};

to get an array's length, use length function - int x = pets.length;

JVM stands for Java Virtual Machine.

Typically you will write three things when you create a new class - prepcode, testcode and real (Java) code.

Use the prepcode to help design the code.

We use for loops instead of while loop if we know how times we want to repeat our loop.

Use pre/post increment operator to add 1 to a variable - "x++"

Use pre/post decrement operator to substract 1 to a variable - "x--"

Chapter-2 - Java

Object-oriented programming lets us extend a program without having to to touch previously tested, working code.

All Java code is defined in a class.

A Class describes how to make an object of that class type. A class is like a blueprint.

Things that an object knows about itself are called instance variables. They represent the state of an object.

Things that an object does are called methods. They represent the behavior of an object.

Classes help us jot down objects that have similar attributes and methods.

The "Shape" class is the superclass, while circles, Rectangle, Triangle and Amoeba are subclasses which inherit all methods from the superclass.

methods

Note - Subclass can be overriden. (Python's ABC) → but easier

A Class is not an object, it is more like an object constructor.

When you create a class, you may also want to create a test class which you'll use to create objects of your new class type.

The test-class is where we put the main() method. The only job of this class is to try out the methods and variables.

Note - Convention of naming test class is <class name> Test Drive.

↑

actual class.

To make a new object use syntax - Dog d = new Dog ();

Dog → class

How to create a complete class —

① Write a class -

```
class Dog {
    int size;
    String breed;
    String name;

    void bark() {
        System.out.println("RUFF! RUFF!");
    }
}
```

Instance variables

method

② Write a tester class.

```
class DogTestDrive {
    Public static void main (String[]
                              args){

    // Dog test code goes here

    }
}
```

③ In your tester, make an object and access the object's variables and methods.

```
Class DogTestDrive {
    Public static void main (String[] args) {
    Dog d = new Dog (); ← make a Dog object
    d.size = 40; ← set the size using (.) operator.
    d.bark (); ← and to call its bark method.
    }
}
```

We have to use "new" keyword to create an object in Java, unlike Python where we use d = Dog ()

4. The two functions of main are- 1) to test our real class
2) to launch our Java application.

A real Java application is nothing but objects talking to each other.
↑ calling methods.

(||) is the OR operator.

2.2.12

Variables - Java

Java cares about type. It is not like Python, which uses duck typing.

There are two types of variables - Primitive and Object Reference.

Primitive variables - hold fundamental values.

Object variables - holds references to objects.

To define a variable we must define its type and name.

↑ Not required in Python.

Variables should be thought of as cups, that we need to know the type of before we use our container.

The sizes of six primitives in Java are - byte, short, int, float, long, double
8 16 32 32 64 64.

Note - when we define a float, we use "f" with our value assignment, so that it is not considered as a double. Example - float a = 32.25f.

Literal value types - int, char, double, boolean.

The rules to variable - naming are —

1) It must start with a letter, underscore or a dollar sign, but not a number.
2) We can use numbers after the first character.
3) Do not use reserved words.

Object variables remember the address to the object.

If a variable is marked final, it will not be able to be assigned to any other value.

A variable can have no value at all - "Null". Python's - "None".

How to declare an array- `int [] nums;`

`nums = new int [2]` ← size

`nums [0] = 2;`

`nums [1] = 1;` 1:11 `nums [6] = 23`

Array are objects.

When we have object Dog in a Dog array we have to use array notation to use dogs functions. Example - `myDogs [0].name = "Fred";`

There is no such thing as a primitive array. Arrays holds primitives.

How objects behave

- We can pass values to our methods. Also known as arguments/parameters.
- A Method user parameters. A caller passes arguments.
- If the method takes a parameter we must pass it something of appropriate type.
- We use return function to get things back from a method.
  If we declare a method to return a value, you must return a value of the declared type.
- Values passed in and out of methods can be implicitly promoted to a larger type or explicitly cast to a smaller type.
- The value you pass as an argument to a method can be a literal value or a variable declared parameter
  =32
- A method must declare a return type. A void return type means the method does not return anything.
- If a non-void return type is declared, a value must be returned.
- Getters and setters let us get/set values to return types.
- By forcing everyone to call a setter method, we can protect the cat from unacceptable size changes and increase security.
- Mark instance variables as private, and getters and setters to public to increase security.

Extra- Strength Methods

To-DO for developing a class-

- ☐ Figure out what the class is supposed to do.
- ☐ List the instance variables and methods.
- ☐ Write prepcode for the methods.
- ☐ Write testcode for the methods.
- ☐ Implement the class
- ☐ test the methods.
- ☐ Debug and reimpliment as needed.

Prep Code - A form of Pseudo code, to help us focus on the logic and not syntax.

Test code - A class on methods that will test the real code and validate that it's doing the right thing.

Real code - The actual implementation of the class

Most Prep code includes three parts: instance variable declarations, method declarations, method logic. Method logic is the most important part of prep code. It defines what needs to happed.

Check Yourself Method - Public String checkYourself (String guess) {

int guess = Integer.parseInt (String Guess);

↑ Converts a string to int

String result = "miss";

type in

for (int cell : location Cells) { ... .

it array

same as for i in location cells, except we have to mention type.

Use the prep code to design the test code.

Math.random returns a float number from 0-1.999, hence we use a cast to turn it into an int.

① Make a random number -

```
int randomNum = (int) (Math.random() * 5);
```

↑ cast operation.

↑ As we need a number in the range (0-4))

② Let user input -

```
string guess = helper.getUserInput ("enter a number");
```

Two types of for loop -

① Regular - for (int i=0; i<100; i++) { } ← loop body.

↑ initialization ↑ boolean test ↑ iteration expression

② Enhanced - for (String name : nameArray) { }

↑ The elements in the array must be of this type.

↑ with each iteration a different array will assigned to "name" (same as "i")

↑ The collection of items we want to iterate over

Final code for Simple Dot Com and Simple Dot Com Tester -

```
Public ~~static~~ class Simple Dot Com Test Drive {

    Public static void main (String [] args) {
        simple Dot Com dot = new Simple Dot Com ();
        int [] locations = {2, 3, 4};
        dot.set location Cellss (locations);
        String UserGuess = "2";
        String result = dot.checkYourself (UserGuess);
        }
~~Dot~~ }

Public class Simple Dot Com {
    int [] location Cells;
    int numOf Hits = 0;

    Public void setLocation Cells (int [] locs) {
        location Cells = locs;
    }

    Public string checkYourself (String stringGuess) {
        int guess = Integer.parseInt (stringGuess);
        string result = "miss";
        for (int cell : location cells) {
        if (Guess == cell) {
            result = "hit";
            numberOf Hits ++;
            break;
            }
        }
```

Note - Bug - A guessed number is not popped off, hence can be guessed repeatedly to sink a ship.

```
if (num of Hits == location Cells.length) {
    result = "kill";
} }
System.out.println (result);
return result;
```

Using the Java Library

Just like Python, Java has its own library containing modules.

With Java.arrays are just like python's, they can't be mutated.
Hence we use ArrayLists. (Python's List)

ArrayList's methods - add ()
remove ()
contain ()
isEmpty ()
indexOf ()
size ()
get ()

Returns true if there's a match for object parameter.

Python's

| append ()
| pop ()
| in syntax
| new syntax
| ~~new syntax~~
| length ()

An arrays size has to be declared beforehand.

To add an object in an array, we must assign a specific place for it.

Array -
myList[1] = b;

ArrayList -
mylist.add(1)

ArrayLists in Java 5.0 are parameterized - ArrayList<String>.
meaning we are declaring the type of objects to be put inside the list.

ArrayLists can be heterogenous. Simply don't use parameters.

To use ArrayLists, import it first using - import java.util.ArrayList;

And operation - &&
OR " - ||

Not Equals = != and !

Better Living in objectville

In chapter 2 we created a class "shape" from which "circle", "triangle" all inherited the rotate() method. "shape" is the superclass, and "circle"... are the subclasses.

Inheritence allows us to jot ~~and~~ down our classes and to help cleaner code.

Overriding means that the subclass redefines one of its inherited methods when it needs to change or extend the behavior of that method.

An inheritance relationship means that the subclass inherits the members of the superclass. meaning - instance variables and methods.

Instance variables are not overriden as it is not necessary because they don't define any special behaviours. A subclass can choose its own instance variable values.

Basic Inheritance code - Public class Doctor {

```
    boolean worksAtHospital;
    void treatPatient() {
    // perform checkup
    }
}
Public class Surgeon extends Doctor {
    void treatPatient() {
    // Perform surgery
}
```

12/12/24

When creating superclass and subclasses, identify the instance variables and methods first.

Always look for opportunity to use abstraction by finding two or more subclasses that has common behaivor.

Use the dot operator to call methods on objects. (same as Python)

IS-A, HAS-A tests is a good way to test our class-map.

It is possible to get our overriden method to do the basic method function and more by extending it. We use super.method name(); when we want this.

There are four access levels = Private, Default, Protected and Public.

Access levels control who sees what.

Public members are inherited.
Private members are not inherited.

With Polymorphism, the reference type can be a superclass of the actual object type. Example- Animal myDog = new Dog();

We can have polymorphic arguments and return types.

With Polymorphism, if we create a vet class and pass animal class as a parameter, the vet will be able to handle any type of animal.

Rules for overriding a method - 1) Arguments must be the same and return types must be compatible.

2) The method can't be less accessible

An overloaded method is nothing more than having two methods with the same name but different arguments lists.

Note - It has nothing to do with inheritance or polymorphism.

Overloading facts - ① The return types can be different.

② You can't change only the return type. The argument lists must be different too.

③ We can vary the access levels in any direction.

Use "@override" before the subclass method to override superclass's methods. Using it makes our code cleaner.

Polymorphism helps us with overloading and overriding.

Polymorphism means "Many" + "Behavior" for the same function.

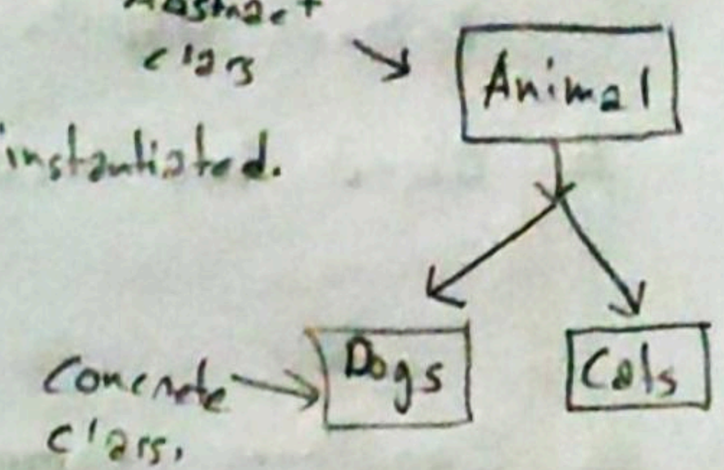
For overloading, Polymorphism lets us have different behaviours according to our input. (Create ~~one~~ multi ways of same function)

For overriding, Polymorphism lets us inherit and use different functions.

Serious Polymorphism — Asbstract classes and Interfaces

When we mark a class as "abstract, the compiler will not let us create an istance of that type.

For example - We do not want to create an instant variable "animal" using animal class simply because it does not make sense and raises a lot of questions.



Concrete classes are those specific enough to be instantiated.

To mark a class as abstract, put the keyword abstract before class declaration. → abstract class Canine extends Animal { }

An Abstract class has virtually no use, no value. unless it is extended.

When we mark a method as abstract, it means we must override the method. (This is what we do in Python) → a abstract method

An abstract method has no body. ex- Public abstract void eat(); → no body.

If we mark a method abstract, we must mark the class as abstract too. We can not have an abstract class in a non-abstract class.

We must override the abstract methods as they don't have a body

The first concrete class must override the method. We can choose not to override it if it is an abstract class extending an abstract class.

An Abstract class can have both abstract and non-abstract methods.

Class "Object" is the mother of all classes.

Any class that does not explicitly extend another class, implicitly extends Object.

Object has methods like - equals(), getClass(), hashCode, toString and more.

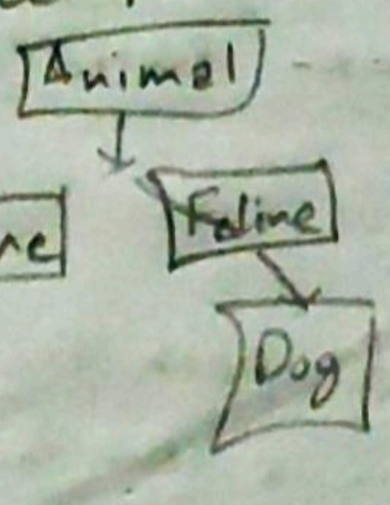
Everything comes out of an ArrayList<object> as a reference of type Object, regardless of the what the object is.

The compiler decides whether you can call a method based on the reference type, not the actual object type.

A Java interface is like a 100% pure Abstract class with having all abstract methods until the first concrete class.

This lets us add pet() methods to only petable animals.

example -



To DEFINE an Interface - Public interface Pet { ... }

↑ use "interface" instead of "class"

To IMPLEMENT an Interface -

Public class Dog extends Canine implements Pet { ... }

Interface methods are implicitly public and abstract.

" " don't have any body

Use an Abstract class when you want to define a <u>template</u> for a group of subclasses, and you have at least some implementation code that all subclasses can use.

Use an interface when you want to define a <u>role</u> that other classes can play, regardless of where those classes are in the inheritance tree.

To use superclass version of a method from subclass that's overriden the method, use the "<u>super</u>" keyword. Ex - super.runReport();

Interfaces are just Hardcore abstract classes, meaning they have no concrete methods. Only Abstract methods.

<u>Similarilarities</u> between Interfaces and Abstract classes - Cannot be instantiated.

— Have abstract classes.

Note - to be instantiated classes must have concrete classes.

Differences - Abstract classes contain at least one abstract methods, while Interface contains only abstract methods.

Life and Death of an Object

In Java, we care about two areas of memory -
  # Heap - the one where objects live.
  # Stack - the one where method invocations and local variables live.

There are two types of variables - Instance variables
                                  - Local / Stack variables.

# Instance Variables are declared inside a class but not inside a method.
  Example - Public class Dark {
              int size; ← Instance Variable.

Local Variables are declared inside a method including method parameters.
They are temporary and only exists as long as the method is running / on stack.
Example - Public void foo (int x) { (Note - Remember Scope? same thing)
            int i = x+3
            boolean b = true; # i and b are local variables.
          }

When we call a method, it lands on the top of a call stack.
The method on the top of the stack is always the currently executing method.

IS the local variable is a reference to an object, only the variable (reference / remote control) goes on the stack.

Instance variable live on the heap, inside the object they belong to.

Every class we create has a constructor. We use it to create instances of the class. Use 'new' keyword to use the constructor.

Creating a constructor - Public class Duck {

```
    Public Duck() {
        System.out.println("Quack");
    }
}
```

← Construction code

The construction gives us a chance to step into the middle of new.

Methods must have a return type, but constructors cannot have a return type.

If we have more than one constructors in a class, it means we have overloaded constructors.

If you write a construction that takes arguments, and you still want a no-arg Construction, you'll have to build the no-arg construction yourself.

If you have more than constructors in a class, they must have different argument lists.

All the constructors in an inheritance tree must run when you make a new object.

~~To invoke a~~ Note - First the subclass construction gets invoked, ~~but~~ which invokes the superclass, but the superclasses construction finishes before subclass.

To invoke a superclass construction, use "super();" in subclass construction
Note - If we don't do it, compiler does it for us.

The superclass parts of an object must be fully formed before subclass parts can be constructed.

The call to "super();" must be the first thing in each construction.

Constructors can accept arguments.

Use "this ()" to call a constructor from another overloaded constructor in the same class.

Just like "super()", "this" must be the first statement in a constructor. Hence, we can't use both.

Note- It's so much easier to use default values in python. use (value=24)

Numben Mattens

Methods in the Math class don't use any instance variable values. And because the methods are 'static', you don't need to have an istance of Math.
All I need is the class and it's methods.
Method exampler of Math - min (), max (), abs (), round ().

We call static methods using class name.- Math.min (88, 86);

We call non-static methods using reference variable name - song t2 = new Song();
t2.play ()

It is impossible to instantiate abstract classes.

We can restrict non-abstract classes from instantiation by marking the constructon "private".

A method marked private means that only code within the class can invoke the method. It is the same for constructons.

Math class's constructon is private, hence we can't create an instance of it.

Static methods can't use (non-static) instance variables on non-static methods.

Static variables are not set to zero each time while incrementing, whereas instance variables ~~can~~ may be set to zero.

Static variables are shared. :- one/class.

All instances of the same class share a single copy of static variables.

Instance variables - 1/instance.

All static variables in a class are initialized before any object of that class can be created.

A variable marked final can never change, and hence is a constant.
Ex- PI in Math.

Variables are marked "Public" so that any code can access it.

Constant variable names should be in all caps.

A "final" method means you can't override the method.

A "final" class means you can't extend (create a subclass) of it.

When you need to treat a primitive like an object, wrap it.

Since Java 5.0, we do not need to wrap and unwrap before inputting such primitives in an array. Java does it willingly. This feature is called Auto-boxing.

We can turn a primitive int or a ~~float~~ Double. from string using parseInt/Float Double.

We can also turn a string a primitive number to string using -

1) concatenate it — double d = 42.5
String doubleString = "" + d;

2) double d = 42.5;
String doubleString = Double.toString (d);

We can use formatting to output in a specified way.

"%" in string formatting is the place holder.

Example - format ("I have %,.2f bugs to fix.", 976578.09876);
↑ format specifiers ↑ Argument to be formatted

output - I have 976,578.10 bugs to fix.

Note - Using a comma in format specifier resulted in comas in results.

A format Specifier can have five different parts. (not including the "%")

- % [Argument number] [flags] [width] [.precision] type

↑ These are for special formatting like inserting commas and such.

↑ This defines the minimum number of characters that will be used.

Type is the only non-optional part.

Date formats use a two-character type that starts with "t".

We use - String.format("%tc", new Date()); for complete date and time. Output - Sun Nov 28 14:52:41 MST 2004.

For just the time - %tr,

input - String.format("%tr", new Date());

Output - 03:01:47 PM

Day of the week, month and day - %tA, %tB, %td

month day

month week

day of the month

- Date today = new Date();

String.format("tA, %tB %tD", today, today, today)

- Sunday, November 28

For a time-stamp of "Now", use Date. But for everything else use Calendar.

Syntax - Java.util.Calendar.