Java Programming Basics

Java Keywords : reserved words, 51 in total

Variables : combination of a location in computer memory and an associated name

Eg : `int age = 42;`

Static Binding :
- data type bound to variable
- data type for variable cannot be changed
- Java is statically typed :

```
int age = 42;
```

Dynamic Binding :
- data type bound to value
- data type of variable can be changed
- Python

```
age = 42
```

Primitives vs Objects

Primitive :
- simply a value, by itself, no additional data

Reference :
- value that refers to an object stored in another location

8 primitive types : byte, short, int, long, float, double, boolean, char

Object types don't have a fixed memory allocation size

Typecasting

Eg: int intNum = 3
double doubleNum = intNum;

Manual Type Casting
- convert larger type into smaller type
- convert one object type into another

Eg: double doubleNum = 3.5;
int intNum = (int) doubleNum;

Truncation
- loss of precision from one type to another
- occurs when manually type casting to convert larger to smaller type
- occurs because there isn't enough space

Methods

Eg: public double getArea (double length, double width) {
double area = length * width;
return area;
}

For path: Name, parameter, method body, return type

Method vs Function
- function is any block of reusable code
- method attached to class or object

Stack vs Heap

Stack:

- store primitive and object references
- items get added and removed as given method executes
- LIFO
- items maintained as long as related method is running
- items can only be accessed by related method

Heap:

- store object themselves
- object stay until application is done
- object accessible from anywhere in the program

Access Modifiers

- determine how other classes are allowed to access your variables and methods

Public:

- class can be accessed from anywhere

Private:

- only the defining class can access data

Protected:

- access is restricted to defining class, package or subclass

Default:

- access is restricted to defining class or package

Summary

Access	Inside Class	Inside Package	Outside package by subclass	Outside package
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

Arrays

- o fixed-sized data structure that is used to store multiple values

Eg: `int[] numbers = {1, 2, 3, 4};`

or

`int[] numbers = new int[4];`

Loops

- three types : while, for, do while

Eg of while: `while (condition) { }`

Eg of for: `for (int i = 0; i<5; i++) { }`

Eg of do-while: `do { } while (condition);`

JavaDoc

- documentation generator that produces a searchable HTML document defining classes and interfaces of an application

Eg:

```
public class JavaDocEg1 {
    /**
     *
     * @param string
     * @param num
     * @return
     */
    public String sayMethod(String str, int num){
        return "result";
    }
}
```

Defining Classes

What is an object?

Benefits of OOP approach:
- better software reliability
- better maintainability
- reduced cost of developing software

An object is a data structure that encapsulates two things:
- state of object
- behavior of object

Classes vs Objects

- class is like a blueprint for a kind of object
- class defines state and behavior that an object of that class will have
- a single class can instantiate multiple objects of the same type
- each object is considered an instance of the class

Defining a Class

Class Definition: class name, class variable, constructor, methods

Instance vs Class Variable:
- Instance: state variable that have unique value for each object
- Class Variable: state variable that belong to the class itself, and are the same for every object.
  static keyword identifies that variable as belonging to the class (not individual object)

final keyword: make item non-changeable

Working with Objects and Singleton Class

- singleton class meaning only one class instance in JVM

Eg:

```
public class Server {

    private static Server reference = new Server();
    public ArrayList hotelRooms;

    // Now private constructor prevents client app from creating
    // Server class instance and it singleton class
    has access to it hotelRooms ArrayList

    private Server() {
        this.hotelRooms = new ArrayList();
    }

    public Server getInstance() {
        return reference;
    }
}
```

Garbage Collection

- check all instantiated objects, and destroys them if they do not have any references

Object - Oriented Programming

Packages

- essentially serves the same purpose as "namespaces"
- package keyword located at the top of a Java file

Inheritance

The act of one class acquiring properties and methods from another class

- want to go from general to specific

The Object SuperClass:

- every class inherit from superclass Object.
- Some methods that all object have:
  - clone()
  - equals()
  - hashCode() -> provide unique hash code for each object
  - toString()

Polymorphism

- ability of an object to take many forms

Abstract Classes

- defines the behavior for each of the subclasses, but we cannot directly instantiate the abstract class itself.
- allow us to create abstract methods.

Eg: public abstract void speak();

Interfaces

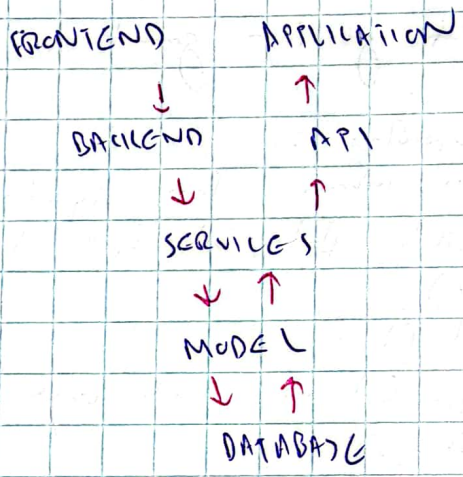
- allow us to avoid hardcoding feature in an application

Interface vs Abstract Class

- Abstract Class:
  - can have class variables
  - can have abstract and concrete methods
  - can have instance variables
  - subclasses can only extend one class

- Interfaces
  - can have data validation
  - every method abstract
  - cannot have instance variable
  - class can implement more than one interface and have multiple inheritance

Layering



Common Types

Exceptions

Error Class and Exception Class:

- error class is used to indicate a serious problem that a application should try NOT to handle
- a exception class is when there is a less catastrophic event that a application should try to handle

Throwable Class:

- error and exception class inherit from abstract class Throwable hence both will contain:
  - type of problem -> either exception or error
  - problem message
  - stack trace when exception occurred

Throwing an exception

- when an error occurs within a method, it creates an exception object

Checked Exceptions:
- known to a compiler
- exception must be handled

Unchecked Exceptions
- unknown to compiler
- also referred to as runtime exceptions
- used when we expect caller of method could recover from exception

Handling Exceptions

Eg:

```
try {
    read();
}
catch ( FileNotFoundException e ) {
    e.getLocalizedMessage();
}
finally { }
```

Enums

- a special type of constant that allow a variable to be set from an enumerated list

An Enum is a Class

- declaration of an Enum defines a class
- This class can exist with another class or as a standalone class

Eg:
```
enum Steplight {
    RED,
    YELLOW,
    GREEN
}
```

Assigning:
```
Steplight mySteplight = Steplight.RED;
```

Scanner

- Scanner Class can read and parse simple text.
  - parses primitive types and string types into tokens
  - default delimiter is whitespace

Eg:
```
Scanner scanner = new Scanner(System.in);
```

Eg:
```
Scanner scanner = new Scanner("This is a line");
System.out.println(scanner.nextLine());
```
Output: text This is a line

Eg:
```
Scanner scanner = new Scanner("This is a line");
while (scanner.hasNext()) {
    System.out.println(scanner.next());
}
```

output:
```
This
is
a
line
```

Dates and Calendar

The Date Class:
- represents a specific instance in time

Eg: `Date dd = new Date();`

The Calendar class:
- abstract class provides methods for manipulating dd and time.

Eg: `Calendar calendar = Calendar.getInstance();`

RegEx
- Used to match or find Strings
- In Java contains 3 classes to support these operations:
  - Pattern
  - Matcher
  - PatternSyntaxException

How to do these many things:
- create a pattern based on specialized syntax
- use matcher to determine if pattern exist in string provided

Eg:
```
import java.util.regex.Pattern;
private final String emailRegex =
        "^(.+)@(.+).com$";

private final Pattern pattn = Pattern.compile(emailRegex);
```

String Methods
- A String pool is a way of storing only one copy of a string
- Strings in Java are immutable, meaning they cannot be changed after creation

Generics and Collections

Generics

- Benefits :
  - stronger type checks at compile time
  - remove need to cast object
  - allow developers to implement generic algorithms

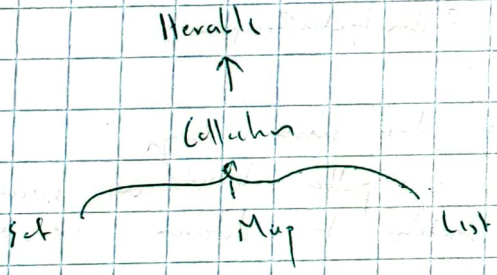
Eg : creating a list of strings :

```
List strings = new List();
```

To identify what types are stored in the list :

```
List<String> strings = new ArrayList<String>();
```

Collections



- Iterable interface defines single method called iterator()
- collection framework consists of several data structure like : Set, Map and List. In addition, there are utility classes like collections and Arrays that provide methods for sorting and creating empty lists

Eg : Note -> creating instance of List but using ArrayList to instantiate it. Because list is an interface

```
List<String> mylist = new ArrayList<String>();

mylist.add("am");
mylist.add("the");
mylist.add("the");

mylist.remove("am");
```

```
for ( String name : myList ) {
        System.out.println ( name );
}
```

Sorting Collections

Collections framework provides a sort method to sort lists of following types:

- Strings
- Wrapper objects
- User-defined classes

Eg: `List<String> names = new LinkedList<String>();`
`Collections.sort(names);`

Wrapper objects: wraps primitive variable into object example int to Integer.

User-Defined Classes: how to implement Comparable interface.
- comparable interface provides a method compareTo.

Eg: `class person implements Comparable<Person> {`

`public int compareTo ( Person pers ) {`
`return name.compareTo(pers.name);`

Advanced Collections

Maps

- normally required search time grows linearly with number of objects, O(n)
- With maps, constant lookup time - O(1), use key-value pairs

Map is an interface:
- list of keys
- list of values
- set of key-value mappings

Eg: Map<String, Person> mapOfPeople = new HashMap<String, Person>();

Person milk = new Person("Milk", "milk@gmail.com");
mapOfPeople.put(milk.getEmail, milk);

mapOfPeople.get("milk@gmail.com");

for (String email : mapOfPeople.keySet()) {
    System.out.println(email);
}

Sets

A set is a collection type that has no duplicate values

Three concrete implementations:

- HashSet: uses a hash table to store elements, provides constant-time performance for basic operations such as add, remove and contains
- TreeSet: uses a tree structure to store elements, keeping them in sorted order.
- LinkedHashSet: similar to HashSet but maintains a linked list of the elements in the order in which they were inserted

Eg: Set<String> mySet = new HashSet<String>();
mySet.add("Hello");

Retrieving objects from a set: have to iterate over the set

Eg: String foundObject;

for (String text : mySet) {
    if (text.equals("Hello")) {
        foundObject = text;
    }
}

Queues

Allows FIFO order processing

- common use for Queue is managing asynchronous processing

Eg: `Queue<String> myQueue = new LinkedList<String>();`
`myQueue.add("Hi");`
`myQueue.add("There");`

"poll" method retrieves first element and removes it.

Eg:
```
while (!myQueue.isEmpty()) {
    System.out.println(myQueue.poll());
}
```

Overriding hashcode and equals

Eg:
```
public class People {
    private String address;
    public String name;
    public String mail;
}
```

Mail equal method to sort by address:

```
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    People people = (People) o;
    return address.equals(people.address);
}
```

Overriding H hashcode:

```
public int hashCode() {
    return Objects.hash(address);
}
```

Hotel Reservation Application

Architecture :

UI COMPONENTS	RESOURCES	SERVICES	MODELS
			FREEROOM
MAIN MENU	HOTEL RESERVATION RESOURCE	RESERVATION SERVICE	ROOM
ADMIN MENU	ADMIN RESOURCE	Custom Service	RESERVATION
			CUSTOMER

* Layering : ensure there are no cross communication calls from one layer to another