 (866) 886-9798 • AskBenefits@allegisgroup.com

rp-globalprotect.allegisgroup.com

^^ portal ^^

Perks- sarcastic1

CERTIFICATIONS- https://education.oracle.com/oracle-certification-paths-all

Governance- the way we ensure quality, consistency, and predictability throughout the different stages of software development

-establishes chains of responsibility, authority, and communication

Robust, enabling, lightweight, transparent, flexible framework

----------SET UP TDD---------------

New directory

Right Click- Mark directory as Test

File -> Project Structure -> Libraries -> Add Maven -> junit

Sonarqube – testing tool for bugs

--------------------------------------------------

args – keyword

can run resulting program in command line

public static void main(String[] args){

String first = args[0]

String second = args[1];

$ javac Main.java

$ java –cp . Main

$ first second

String number = “2”;

int num = Integer.parseInt(number);

num = 2

varargs

public void someClass(int… args){

variable amount of args

}

JShell

read-evaluate-print-loop

$ jshell

$ /help

$ /exit

scratch variable (like count) - $2 + $1 (1st operation plus second operation)

snippets- variables, expressions, classes, IF, Loop, imports

/list, /edit, /drop, /save, /open, etc

/list –all , /list –start, /list 4 5 (lines)

/set editor “C:\\Path” + /edit –start (opens snippet in editor)

/drop 7 8 (lines), /drop x (variable)

/save C:\Users\sf\Documents\snippets.txt

/save –history or –all or –start C:\Users\etc

/open C:\Users\Devon\notes.txt

/open JAVASE (see all packages) /open PRINTING (print methods)

/vars (all variables) /vars –all (even dropped vars)

/methods (all functions) /methods –start (if in startup, probably none), /methods helloWord

/types (classes, enums, interfaces) /types Day

/imports

/env –class-path someJar.jar (import classes from jar)

$ import java.lang.Exception

--Benefits of JShell--

quick feedback, experiment with unfamiliar code (foreign class, Java APi, 3rd party library, etc)

bypass and preserve the existing program (run quick tests, simulate a scenario)

test ideas on how to build your program (start with simple tests, build up, eventually integrate)

need java platform/ right click on project or going to tools and open java platform

static blocks (or static clause)- used for static initialization of a class. the code inside is executed only once, when the class is first loaded into memory (executed before anything else including constructor)

static {

} (above constructor)

Constants-

declared using ‘final’ keyword

compiler will give error if attempt is made to change constant

Access Control

* hide fields and methods from other classes
* determine how internal data gets changed
* keep the implementation separate from the public interface

Encapsulation

* hiding object fields
* uses access control to hide the fields
* safe access provided by getter and setter methods
* in setter methods, use code to ensure values are valid
* Encapsulation mandates programming to the interface
* a method can change the data type to match the field
* a class can be changed as long as the interface remains the same
* encourages good Object Oriented Design

String equality

string.equalsIgnoreCase(string2) = .equals while ignoring uppercase

string.compareTo(string2) compares two strings lexicographically. each character is given Unicode value

returns 0 if strings have equal value, less than 0 if string has less characters than string2, more than 0 if string has more characters

StringBuilder has better performance than String for concatenation

stringBuilder.insert(3, “word”) – inserts “word” at position 3

string.split(“ “) – splits string into an array of strings at each space

string2.split(“-“) –splits string into array of strings at each hyphen

string.split(“\\s+”) – splits at any whitespace (new line or tab or space)

string.split(“\\D+”) – splits at any non-digit char

string.isEmpty - returns Boolean if no length

string.replace(“the”, “a”) – replaces **all** instances of ‘the’ with ‘a’

string.replaceAll(“\\p{Punct}”, “”) – replaces all punctuation with whitespace //FOR REGEX

Data Types

Integral- bye, short, int, long

Floating- float (must use F), double (default)

textual type- char (must use ‘ )

logical- Boolean

a = a/b -equals a /= b

a = a % b equals a %= b

2D arrays

String twoD[][] = new String[5][3] //5 by 3 array aka 15

String Monday[] = {“apple”, “carrot”, “beans”}

twoD[0] = Monday

twoD[4][0] = Monday[0]

for (int i = 0; i < 5; i++){

loops through 1st dimension aka 5

for (int j etc.. loops throw 2nd length aka 3

Enums-

enum Month { JANUARY, FEBRUARY, MARCH, etc) should be in caps since it’s a constant,

enum is a special data type for defining a preset amount of constants that are only possible values

overriding- (public via private) children can’t have more restrictive access than parent method they are overriding

Parent asdf = new Child();

asdf instanceof Child //returns Boolean

abstract – cant be instantiated, can have concrete properties, methods, and abstract methods

modular design

directory structure layout- how packages and files are organized

package that a class belongs to is defined in the source code

$ usr/lib/etc/Project.jar exampleproject.ExampleProject

^^specifies the path and JAR file, then the package name, then the class name of Main method

Client/Server Two-Tier Architecture

* client/server computing involves two or more computers sharing tasks
* each computer performs logic appropriate to its design and stated function
* front-end client communications with the back-end database
* client requests data from the back end
* server returns the appropriate results
* client handles and displays data
* performance penalty

Client/Server Three-Tier Architecture

* three-tier client/server is a more complex, flexible approach
  + data
  + business logic
  + presentation
* each tier can be replaced by a different implementation
  + data tier is an encapsulation of all existing data sources
  + business logic defines business rules
  + presentation can be GUI, web, smartphone, or even console
* data outputs to business logic which outputs to presentation

Java Modular Design Principles

* core feature is its ability to reuse code
  + large programs can be built from small programs
  + basic unit of reuses has been a class (programs are classes)
  + Java promotes inheritance for reusing behavior and interfaces for reusing abstractions
* packages are grouped in JARs and JARs are the unit of distribution
* large Java codebases (distributed in many JARs) can be difficult to control which classes and interfaces are reusing code
* only way to share code between packages is through the public modifier but then the code is shared with everyone
* a module is a set of packages designed for reuse (since JDK 9)
  + modularity adds a higher level of aggregation above packages
  + a module is a reusable group of related packages, as well as resources (like images and XML files) and a module descriptor (programs are modules)
* made up of (a set of )
  + exported packages – intended for use by code outside the module
  + concealed packages – internal to the module

Module System

* addresses **reliability, maintainability, and security**
* supports creation of applications that can be scaled for small computing devices
* addresses dependency issues and provides encapsulation at a large level
* is useable at all levels- Applications, Libraries, and the JDK
  + Java Language & JVM (bottom to top)
  + Modular JDK
  + Modular Libraries
  + Modular Applications (Module Aware Tools)
* addresses the following issues at the unit of distribution/reuse level
  + dependencies
  + encapsulation
  + interfaces
* is a full-fledged Java component
* explicitly declares
  + dependencies on other modules
  + what packages it makes available to other modules
  + only the public interfaces in available packages are visible outside the module
* modules can’t access each other by default (even classes declared public)
* readability is whether a class is accessible to outside code
  + the module-info classes must be edited to specify how modules read from each other

Create Modular App

* module-info.java
* module Name { requires asdf; exports demo;}
* transitivity allows for readability up the requirement chain
* Modular JDK
* before JDK 9, JDK was huge and monolithic which increased-
  + download time
  + startup time
  + memory footprint
* JDK now consists of over 90 modules
* each module is a well-defined piece of functionality of the JDK
* now its- more scalable to small devices
* improves security and maintainability
* improves application performance
* Java SE Modules are classified into 2 categories
  + standard modules (java.\* prefix)
  + jdk specific- modules not defined in the Java SE 9 platform (jdk.\*)
* base module is java.base, all other modules depend on base