**Chapter 1: Into to Computers and Programming:**

Terabye(TB) 🡪 1 trillion (1012) bytes

Gigabyte(GB) 🡪 1 billion (109) bytes

Metabyte(MB) 🡪 1 million (106) bytes

Kilobyte(KB) 🡪 1 thousand (103) bytes

RAM (random access memory) and Cache (much fast, more expensive memory)

byte – 1, short – 2, int – 4, long – 8, float – 4, double – 8, char – 2, Boolean – 1

**Chapter 2: A First Java Program:**

Structure of a Class Definition:

public class ClassName

{

instance field definitions

constructor definitions

method definitions }

Constructor Definitions:

ClassName variable = new className(parameters)

**Chapter 3: Implementing and Testing the Player Class under BlueJ:**

Method Calls:

ClassName.objectname.methodname();

The main() Method:

public static void main(){

Assert Euqals():

assertEquals(methodName(), expected value, tolerance);

**Chapter 5: Variables and Their Definitions:**

Declaring Variables and Legal Names:

type name = initional\_value;

Legal names: uppercase/lowercase letter, digits 0-9, underscore, dollar signs

**Chapter 6: Assignment Statements and Expressions:**

Order of expressions:

Parenthesis, unary + and -, \* / and %, binary + and -.

Type Casting:

z = (double) x / y;

Math Shorthands and Methods:

+=, -=, \*=, /=, %=

math.abs(), sin(), cos(), sqrt(), cbrt(), exp(), log(), log10(), max(a, b), min(), pow(a, b)

String Methods:

Declaration: String name = “Denbigh Starkey”;

Methods: variable.length(), variable.compareTo(), variable.contain(), endsWith(), indexOf(), replace(), to UpperCase();

**Chapther 7: Extending The Player Class:**

Instance Fields vs. Class Fields:

Instance field declared in a class leads to every instance (object) of that class having its own independent copy of the firled. Class fields have only one copy.

Instance field initialized in constructor, Class fields initialized when declared.

If a method is based on the value of a static variable, it is related to the class, not an object, and is therefore static.

**Chapter 9: Conditional Statements:**

*if (condition)*

*statement;*

*else if*

*statement;*

*else*

*statement;*

switch (object of statement)

{

case value: object = newValue; break;

default: system.out.println() }

Comparing Numbers for Equality:

==, !=, <=, >=, variable.equals(“value”);

Logical Boolean Operators:

!, &&, ||

**Chapter 10: Taking Control of Input/Output:**

System.out.format(“output phrase”, variable);

Major Codes for format:

b – Boolean, s – String, c – Character, d – decimal integer, f – decimal number, n

Instancing and Using a Scanner Object:

java.util.\*;

Scanner in = new Scanner(System.in);

System.out.println(“Enter integer value”);

n = in.nextInt();

next() – grabs next chunk of text

JOptionPane

import javax.swing.\*

JOptionPane inBox = new JOptionPane(), outbox = new JOptionPane()

inString = inBox.showInputDialog(“Enter radius”);

radius = Double.parseDouble(inString);

area = Math.PI \* radius \* radius;

outbox.showMessageDialog(null, System.out.format(“Radius: %.2f, area: %.2f”, radius, area);

System.exit(0);

**Chapter 11 - Looping Statements:**

while (condition)

do stuff to variables

*for (initialize; loopCondition; modifier)*

*sum += i;*

do{

code }

while (condition)

**Chapter 12 – 1D Arrays:**

Arrays are static and homogeneous.

Initializing:

arrayType[] name = new arrayType[numberOfElements];

int[] intArr = new int[8]

StudentRecord[] cs160 = new StudentRecord[];

**Chapter 13 – Array Lists:**

Dynamic homogeneous collection of objects.

java.util.\*;

Different Calls to Remember:

arrlist.size(), arrlist.get(4), arrlist.set(4), arrlist.add(index, value), arrlist.remove(index);

Declaring:

ArrayList<Class> name = new ArrayList<Class>();

Wrapping Primitive Types Declaration:

ArrayList<Double> doubleList = new ArrayList<Double>();

**Chapter 14 – Multi-Dimensional Arrays:**

Declaring:

arrayType[][] name = new arrayType[3][4];

**Chapter 15 – Recursive Methods**

General form of a recursive method:

if (base condition)

handle the base case;

else

make recursive call(s);

public static int factorial(int n)

{

if (n == 0)

return 1;

else

return n \* factorial(n-1) }

**Chapter 16 – Inheritance:**

subclass Syntax:

public class ClassName extends SuperclassName{

Constructor:

public Class(all parameters of sub and superclass)

{

super(inherited parameters);

variable = in\_variable;

**Chapter 18 – Exceptions and Exception Handling:**

while (badInput)

{

try

{

System.out.format(“Please enter a positive int value:%n”);

arrSize = in.nextInt();

if (arrSize < 2)

throw new MustBePositiveException();

badInput = false;

}

catch (InputMismatchException exception)

{

String badValue = in.next();

System.out.format(“Input must be int, not %s%n”, badValue);

}

catch(MustBePositiveException exception)

{

System.out.format(“%d is not positive, try again.%n”, arrSize);

}

Making one’s own exception class:

public class MustBePositiveException extens RuntimeException

{

public MustBePositiveException()

{

}

public MustBePositiveExcpetion(String message)

{

Super(message)

}

Reading and Writing to Files:

public static void main()

{

final String filename = “/uers/……/demo.txt”;

FileReader infile = new FileReader(filename);

Scanner in = new Scanner(infile);

String next;

while (in.hasNext())

{

next = in.next();

System.out.println(next);

}

in.close();

**Class Example Part 1(from last year’s exam):**

public class SalesTrip

{

private int income,

cost;

private String city,

date;

private static int numTrips = 0;

public SalesTrip(int in\_income, int in\_cost, String in\_city, String in\_date)

{

income = in\_income;

cost = in\_cost;

city = in\_city;

date = in\_date;

numTrips++;

}

public int profitOrLoss()

{

return income – cost;

}

public static int getNumTrips()

{

return numTrips;

}

public String getCity()

{

return city;

}

public void describeTrip()

{

int gain = profitOrLoss();

if (gain > 0)

System.out.format(“The trip to %s on %s had a profit of $%d,%n”, city, date, gain);

else if (gain < 0)

System.out.format(“The trip to %s on %s had a loss of $%d%n”, city date, -gain);

else

System.out.format(“The trip to %s on %s broke even.%n”, city, date);

}

}

**Class Example Part 2**

import java.util.\*

public class TripRecord

{

private ArrayList<SalesTrip> allTrips = new ArrayList<SalesTrip>();

public void addTrip(SalesTrip newTrip)

{

allTrips.add(newTrip);

}

public int cityProfit(String city)

{

int profit = 0;

for (int i = 0; i < allTrips.size(); i++)

{

SalesTrip iThTrip = allTrips.get(i);

if (iThTrip.getCity().equals(city)

profit += iThTrip.profitOrLoss();

}

return profit;

}

public void describeAllTrips()

{

for (int i = 0; i < allTrips.size(); i++)

allTrips.get(i).describeTrip();

}

}