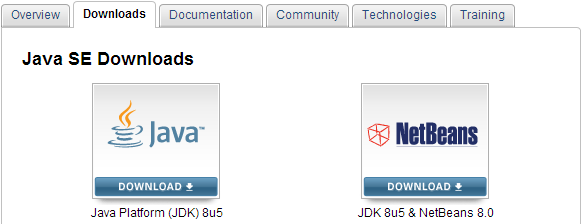
**# 3\_Basic Instructions for JAVA**

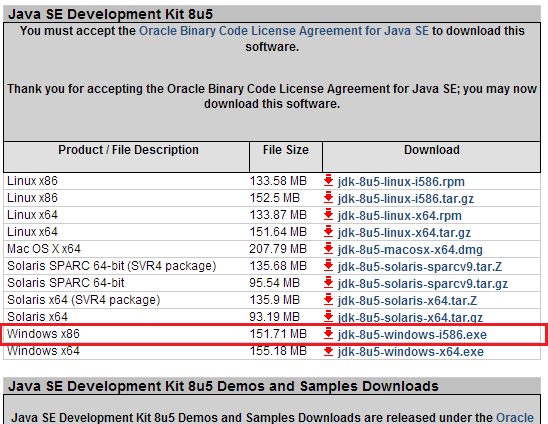
**Created by : Yogendra Patil**

**PhD, University of Alabama**

Google for “Java JDK” -> click on “JAVA SE Downloads” -> click on “Java Platform (JDK) 8u5”



Accept the Oracle Binary Code License Agreement.

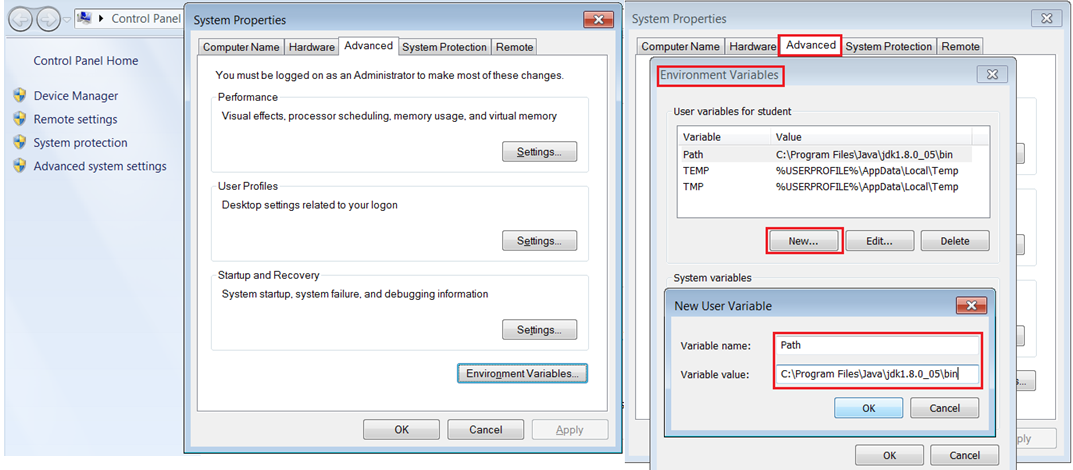


Go on clicking next. After the installation is complete, go to “Start”->”Search for cmd” i.e. open command prompt.

Type in ‘javac’ into the command prompt. You will get an error that says “not recognized”!!!

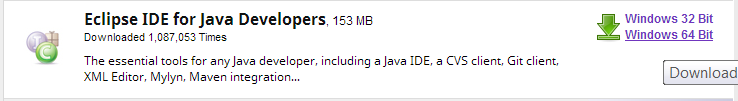
So go to “Start”-> “C: drive”-> “Program Files” -> “Java”-> “jdk1.8.0\_05” -> “bin” and copy the path “C:\Program Files\Java\jdk1.8.0\_05\bin”

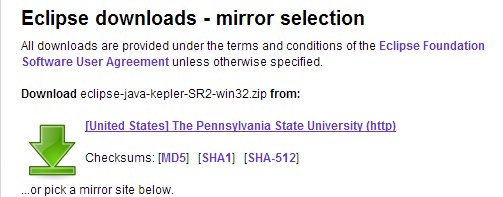
Now go to “Start”->”Computer”-> right click on it and select properties. In properties select option (4th option) on Left Hand Side (LHS) that says “Advanced system settings”. Click on it and select the tab that says “Advanced”. In the tab “Advanced” -> “Environment Variables…” -> “New…” -> Type “Variable name: Path” and type “Variable value: Folder\_path”-> click OK. Finally go to command prompt and type “javac”. You will see bunch of stuff that means that the JAVA COMPILER is set!



Now to Download Eclipse!

Google for Eclipse -> Click on “Eclipse Downloads” -> Find following (Bucky uses 32 Bit even if he has 64 Bit??)

\



Unzip the Eclipse downloaded folder. Copy the folder in “C: Drive”. Open the C: Drive folder of eclipse. Open the item that has “Eclipse Font” (Double click on it)!! Create a workspace in my Documents using “Browse” “C:\Users\student\Documents\Java\_British”. Click OK

Tutorial 1: HELLO WORLD

Part 1: To start a new java project, Start “Eclipse”-> go to File -> New-> Java Project ->(Give your project name) Project name: Tutorial 1 -> click “Next” -> (accept the defaults on next screen) and then click “Finish”.

In the “Package Explorer” tab, right click on “Tutorial 1” -> New -> Class -> (type in your class name under) Name: Application. Now if you want this class to be “public”, click the check box below that says “public static void main…”-> Finish

\*\* type “sysout” then Cntrl + Space to get “System.*out*.println();”

**public** **class** Application {

**public** **static** **void** main(String[] args) {

System.*out*.println("Hello World");

}

}

Tutorial XX: XXXXX

Part 1:

Console:

Tutorial 2: VARIABLES (There are 8 primitive types)

Part 1:

**public** **class** Application {

**public** **static** **void** main(String[] args) {

/// There are 8 types of variables in java

// int - 32bit value in java (for whole number)

//short - 16bit value (short range than int which is 2^32)

**int** myNumber = 99;//initialization

**short** myShort = 847;

**long** myLong = 9797;

//double - floating point number

// short version of double is float

**double** myDouble = 8.57657;

**float** myFloat = 324.3f;

//char- unicode character

**char** myChar = 'y';

//boolean

**boolean** myBoolean = **false**;

**byte** myByte = 127;

System.*out*.println(myNumber);//Console will print "99"

System.*out*.println(myShort);//847

System.*out*.println(myLong);//9797

System.*out*.println(myDouble);

System.*out*.println(myFloat);

System.*out*.println(myChar);

System.*out*.println(myBoolean);

}

}

Console:

847

9797

8.57657

324.3

y

false

Tutorial 3: STRINGS (Non-primitive types)

“Create a new JAVA Project and class”. “But this time don’t check the check box ‘public static void …’, and click finish”.

Now every class must have a main method. So type “main then Cntrl+Space”.

For indentation/formatting select the lines and press “Cntrl+Shift+f”.

Part 1:

**public** **class** Application {

**public** **static** **void** main(String[] args) {

**int** myInt = 7;

String text = "Hello";//Use upper case for String as it is a class and not primitive type like int...

String blank = " ";

String name = "Bob";//String is a class, name is ref, "Bob" - object

String greeting = text + blank + name;

System.*out*.println(greeting);//Only using class refs

//Similarly just using objects

System.*out*.println("Hello"+" "+"Bob");//Only using Objects

System.*out*.println("My integer is:" + myInt);

**double** myDouble = 7.88;

System.*out*.println("My number is: " + myDouble + ".");

}

}

Console:

Hello Bob

Hello Bob

My integer is:7

My number is: 7.88.

Tutorial 4: WHILE LOOP

“Create a new JAVA Project and class”. “But this time don’t check the check box ‘public static void …’, and click finish”.

Now every class must have a main method. So type “main then Cntrl+Space”.

For indentation/formatting select the lines and press “Cntrl+Shift+f”.

Part 1:

**public** **class** Application {

**public** **static** **void** main(String[] args) {

**int** value = 0;

**while**(value <= 10)

{

System.*out*.println("Hello " + value);

Value++;

}

}

}

Console:

Hello 0

Hello 1

Hello 2

Hello 3

Hello 4

Hello 5

Hello 6

Hello 7

Hello 8

Hello 9

Hello 10

Tutorial 5: FOR LOOP

“Create …”

Part 1:

**public** **class** Application {

**public** **static** **void** main(String[] args) {

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

//System.out.println("Value of i is : " + i);

// Another to print output is "printf",i.e print format

System.*out*.printf("Value of i is : %d \n" , i);//printf(string %d,%f, value)

}

}

}

Console:

Value of i is : 0

Value of i is : 1

Value of i is : 2

Value of i is : 3

Value of i is : 4

Tutorial 6: IF

“Create a new …”

Part 1:

**public** **class** Application {

**public** **static** **void** main(String[] args) {

**boolean** cond1 = 1 < 6;

**boolean** cond2 = 5 != 5;

System.*out*.println(cond2);

**int** myInt = 23;

**if**(myInt < 10){

System.*out*.println("Yes its true!");

}

**else** **if**(myInt > 20){

System.*out*.println("Its greater than 20");

}

**else**{

System.*out*.println("no its false!");

}

}

}

Console:

false

Its greater than 20

Part 2:

**public** **class** Application {

**public** **static** **void** main(String[] args) {

**int** loop = 0;

**while**(**true**){

**if**(loop == 5){

**break**;//break out of infinite loop

}

loop++;

System.*out*.println("Looping: "+loop);

}

}

}

Console:

Looping: 1

Looping: 2

Looping: 3

Looping: 4

Looping: 5

Tutorial 7: Getting User Input

“Create …”

In order to import all the necessary class library type “Cntrl+Shift+O”

Part 1:

**import** java.util.Scanner;

**public** **class** Application {

**public** **static** **void** main(String[] args) {

Scanner input = **new** Scanner(System.*in*);//creating a new scanner object

/\*Part 1 for string

System.out.println("Enter your text input: ");//Output prompt

String line = input.nextLine();//Wait for user to enter a line of text

System.out.println("You entered: "+line);\*/

/\*Part 2 for integer

System.out.println("Enter an integer: ");//Output prompt

int value = input.nextInt();// See format

System.out.println("You entered: "+ value);\*/

//Part 3 Floating point

System.*out*.println("Enter your floating point value: ");//Output prompt

**double** value = input.nextDouble(); //See format

System.*out*.println("You entered: "+ value);

}

}

Console:

Enter your floating point value:

78.999

You entered: 78.999

Tutorial 8: DO-WHILE LOOPS, VARIABLE SCOPE

Part 1:

**import** java.util.Scanner;

**public** **class** App {

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.*in*);

/\*

System.out.println("Enter a number: ");

int value = scanner.nextInt();

while(value != 5){

System.out.println("Enter a Number: ");

value = scanner.nextInt();

}

\*/

**int** value = 0;

**do**{

System.*out*.println("Enter a number: ");

value = scanner.nextInt();//only exist in curly brackets not outside

//so cannot use in while(value != 5). So declare outside

}

**while**(value != 5);//checking condn at end of loop

System.*out*.println("Got 5!");

}

}

Console:

Enter a number:

5

Got 5!

Tutorial 9: SWITCH STATEMENTS

Part 1:

**import** java.util.Scanner;

**public** **class** App {

**public** **static** **void** main(String[] args) {

Scanner input = **new** Scanner(System.*in*);

System.*out*.println("PLease enter a command: ");

String text = input.nextLine();

**switch**(text){

**case** "start":

System.*out*.println("Machine Started!");

**break**;

**case** "stop":

System.*out*.println("Machine Stopped!");

**break**;

**default**:

System.*out*.println("Wrong Command!");

}

}

}

Console:

PLease enter a command:

stop

Machine Stoped!

Tutorial 10: ARRAYS

Part 1:

**public** **class** App {

**public** **static** **void** main(String[] args) {

**int** value = 7;//This is a value type not reference type

//You are telling computer to create a memory space (32bit)

**int**[] values;//This is a reference type//Intializing an array

values = **new** **int**[3];//This means point the ref var "values" at [3]

//This allocates a memory to hold 3 integers//Java gives default 0values to each element

values[0] = 10;

values[1] = 30;

values[2] = 40;

System.*out*.println(values[0]);//starts with ZERO

System.*out*.println(values[1]);

System.*out*.println(values[2]);

**for**(**int** i = 0; i < values.length; i++){

System.*out*.println(values[i]);

}

**int**[] numbers = {5,6,7};//Another way to allocate

**for**(**int** i = 0;i < numbers.length;i++){

System.*out*.println(numbers[i]);

}

}

}

Console:

10

30

40

10

30

40

5

6

7

Tutorial 11: STRING ARRAYS

**public** **class** App {

**public** **static** **void** main(String[] args) {

String[] words = **new** String[3];//Initialize of string arrays

words[0] = "Hello";

words[1] = "to";

words[2] = "you";

System.*out*.println(words[2]);

String[] fruits = {"apple","banana","pear","kiwi"};//Another way to initialize

//New Way to Iterate throu and ARRAY !!

//Java will create a temp var fruit of type String and take values from

// the array fruits

**for**(String fruit: fruits){

System.*out*.println(fruit);

}

**int** value = 0;//"int" is a primitive type. Lower case 'i'

String text = **null**;//"String" is not primitive type, its a class!!Upper case 'S'

//This is allocating enought memory to a ref 'text' DMA

System.*out*.println(text);

String[] texts = **new** String[2];//enough memory for two elements

texts[0] = "one";//Complete allocation to "one"

System.*out*.println(texts[0]);

}

}

Console:

apple

banana

pear

kiwi

null

one

Tutorial 12: Multi-Dimensional Array

**public** **class** App {

**public** **static** **void** main(String[] args) {

**int**[] values = {3,5,234};

System.*out*.println(values[2]);

//Multi-Dim can have flexible length

**int**[][] grid = {

{3,5,14},

{2,4},

{1,2,3,4,5}

};//SEMICOLON to close Multi-Dim Array

System.*out*.println(grid[1][1]);

System.*out*.println(grid[0][2]);//Row=0, Col = 2;

String[][] text = **new** String[2][3];

text[0][1] = "Hello There!";

System.*out*.println(text[0][1]);

**for**(**int** row = 0;row < grid.length;row++){

**for**(**int** col = 0;col < grid[row].length;col++){//see the format for getting column length

System.*out*.print(grid[row][col] + "\t");//Use print instead of println

}

System.*out*.println();//Extra line

}

}

}

Console:

234

4

14

Hello There!

3 5 14

2 4

1 2 3 4 5

Tutorial 13&14: Classes + Methods

//To create a class you type keywd "class"

//class is a template to create an object

//objects are variables,and represented in class

**class** Person{

//Instance variables (data or state)

String name;

**int** age;

/\*Class can contain

\* 1.Data:

\* 2.Sub-routines(methods):

\*/

//Sub-routine-methods start with lower case

**void** speak() {

**for**(**int** i=0;i<3; i++){

System.*out*.println("My name is:"+name+" and I'm "+age+" years old");

}

}

**void** sayHello(){

System.*out*.println("Hello There!");

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Person person1 = **new** Person();//creating an object type person

person1.name = "Joe Bloggs";//use 'dot' to refer to variable

person1.age = 37;

person1.speak();//dot->method name -> round brac ()

person1.sayHello();

Person person2 = **new** Person();

person2.name = "Sarah Smith";

person2.age = 20;

person2.speak();

person2.sayHello();

System.*out*.println(person1.name);

}

}

Console:

My name is:Joe Bloggs and I'm 37 years old

My name is:Joe Bloggs and I'm 37 years old

My name is:Joe Bloggs and I'm 37 years old

Hello There!

My name is:Sarah Smith and I'm 20 years old

My name is:Sarah Smith and I'm 20 years old

My name is:Sarah Smith and I'm 20 years old

Hello There!

Joe Bloggs

Tutorial 15: GETTERS and RETURN VALUES

**class** Person{

String name;

**int** age;

**void** speak(){

System.*out*.println("My name is: " + name);

}

/\* Following method does not return anything, so "void"

void calculateYearstoRetirement(){

int yearLeft = 65 - age;

System.out.println(yearLeft);

}\*/

//The following method returns something so "int"

**int** calculateYearstoRetirement(){

**int** yearsLeft = 65 - age;

**return** yearsLeft;

}

//Getters

**int** getAge(){

**return** age;

}

String getName(){

**return** name;

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

Person person1 = **new** Person();

person1.name = "Joe";

person1.age = 25;

//person1.speak();

**int** years = person1.calculateYearstoRetirement();//set to return value of vars

//Just like matlab functions

System.*out*.println("Years till retirements " + years);

**int** age = person1.getAge();

String name = person1.getName();

System.*out*.println("Name is: "+name);

System.*out*.println("Age is:" + age);

}

}

Console:

Years till retirements 40

Name is: Joe

Age is:25

Tutorial 16: Method Parameters

**class** Robot{

//getting something from main class

**public** **void** speak(String text){

System.*out*.println(text);

}

**public** **void** jump(**int** height){

System.*out*.println("Jumping: "+ height);

}

**public** **void** move(String direction, **double** distance){

System.*out*.println("Moving "+ distance + " meters in direction "+direction);

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

// First create the new class type

Robot sam = **new** Robot();

sam.speak("Hi I'm Sam");

sam.jump(7);

sam.move("West", 12.2);

}

}

Console:

Hi I'm Sam

Jumping: 7

Moving 12.2 meters in direction West

Tutorial 17: SETTERS and ‘this’

**class** Frog{

**private** String name;//private- you cannot access outside class

**private** **int** age;//this is encapsulation

**public** **void** setName(String newName){

name = newName;

//Use cmd "this.name = name(...instead of newName)", if parameter passed-name

//is same as class var name!!

}

**public** **void** setAge(**int** newAge){

age = newAge;

}

**public** String getName(){

**return** name;

}

**public** **int** getAge(){

**return** age;

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

Frog frog1 = **new** Frog();

//frog1.name = "Bertie";

//frog1.age = 1;

//Following technique is called encapsulation

frog1.setName("Bertie");//set the instance variable in actual class

frog1.setAge(3);//set the instance variable in actual class

System.*out*.println(frog1.getName());//

System.*out*.println(frog1.getAge());//

}

}

Console:

Bertie

3

Tutorial 18: CONSTRUCTORS

**class** Machine{

**private** String name;

**private** **int** code;

**public** Machine(){

//PART2:Now to call a constr within a constr

//It need to be at first line!!!

**this**("Arnie",0);//Use of kew wd "this"...

//calls Machine('String','int')

System.*out*.println("Constructor running!");

name = "Arnie";

}

**public** Machine(String name){

**this**(name,0);

System.*out*.println("Second Constructor Running");

**this**.name = name;//Use 'this.' when same variable names!!

}

**public** Machine(String name,**int** code){

System.*out*.println("Third Constructor");

**this**.name = name;

**this**.code = code;

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

Machine machine1 = **new** Machine();//This actually causes the method to run and outputs in console

Machine machine2 = **new** Machine("Bertie");

Machine machine3 = **new** Machine("Chalky",7);

}

}

Console:

Third Constructor

Constructor running!

Third Constructor

Second Constructor Running

Third Constructor

Tutorial 19: STATIC VARIABLES & METHODS

**class** Thing{

**public** **final** **static** **int** *LUCKY\_NUMBER* = 7;//'final-JAVA' same as 'const-C++'

**public** String name;//Non-static vars -> Instance vars

**public** **static** String *description*;

**public** **static** **int** *count* = 0;

**public** **int** id ;

**public** Thing(){

id = *count*;

*count*++;

}

**public** **void** showName(){

System.*out*.println("Object id: " +id+ ", "+*description* + ": "+name);//Instance method can access static data!!

}

**public** **static** **void** showInfo(){

System.*out*.println(*description*);//but you cannot reference to a non-static vars (e.g. here name)

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

Thing.*description*= "I am a Thing";

System.*out*.println(Thing.*description*);

Thing.*showInfo*();

System.*out*.println("Before creating objects the count is: "+Thing.*count*);

Thing thing1 = **new** Thing();

Thing thing2 = **new** Thing();

thing1.name = "Bob";

thing2.name = "Sue";

thing1.showName();//You dont have to pass them

thing2.showName();

//Uses of static vars, when you dont want some vars to change, esp in maths like Pi value

System.*out*.println(Thing.*LUCKY\_NUMBER*);

System.*out*.println("After creating objects the count is: "+Thing.*count*);

}

}

Console:

I am a Thing

I am a Thing

Before creating objects the count is: 0

Object id: 0, I am a Thing: Bob

Object id: 1, I am a Thing: Sue

7

After creating objects the count is: 2

Tutorial 20: STRING FORMATTING

**public** **class** App {

**public** **static** **void** main(String[] args) {

//Inefficient coding!! So use String builder class!!

String info = " ";

info += "My name is Bob.";

info += " ";

info += "I am a builder.";

System.*out*.println(info);

//More memory Efficient.

StringBuilder sb = **new** StringBuilder("");//class

sb.append("My name is Sue.");

sb.append(" ");

sb.append("I am a Lion Tamer.");

System.*out*.println(sb.toString());

//Another way to

StringBuilder s = **new** StringBuilder("");

s.append("My name is Roger.")

.append("")

.append("I am a sky driver.");

System.*out*.println(s.toString());

////// Formatting ////////////////

System.*out*.print("Here is a some text.\t That was tab.\n That was a new line ");

System.*out*.print(" Use println for next line.");

System.*out*.printf("\n printFormat: embedd format char, %.2f\n",5.0);

**for**(**int** i = 0; i<11;i++){

System.*out*.printf("%-2d: %s\n",i,"some text here");//%-2d aligns number to left, see after 10!!

}

}

}

Console:

My name is Bob. I am a builder.

My name is Sue. I am a Lion Tamer.

My name is Roger.I am a sky driver.

Here is a some text. That was tab.

That was a new line Use println for next line.

printFormat: embedd format char, 5.00

0 : some text here

1 : some text here

2 : some text here

3 : some text here

4 : some text here

5 : some text here

6 : some text here

7 : some text here

8 : some text here

9 : some text here

10: some text here

Tutorial 21: toString()

**class** Frog{

**private** **int** id;

**public** String name;

**public** Frog(**int** id,String name){

**this**.id = id;

**this**.name = name;

}

**public** String toString(){

//return id + ": " + name;//String concatenation PART 1

/\*

//Another efficient memory saving way is to use StringBuilder();

StringBuilder sb = new StringBuilder();//PART2

sb.append(id).append(":").append(name);

return sb.toString();

\*/

///Best way: to use String.format() method!

**return** String.*format*("%4d : %s", id, name);//PART 3

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

Frog frog1 = **new** Frog(7, "Freddie");

Frog frog2 = **new** Frog(5, "Roger");

System.*out*.println(frog1);//This will look for any toString() method and print it

//But if its not there it will output a hash code e.g.Frog@186db54

System.*out*.println(frog2);

}

}

Console:

7 : Freddie

5 : Roger

Tutorial 22: Inheritance

Main class:

**public** **class** App {

**public** **static** **void** main(String[] args) {

Machine mach1 = **new** Machine();//object for machine method

//this allocates memory for machine

//You can now those methods in Machine to get outputs

mach1.Start();

mach1.Stop();

Car car1 = **new** Car();

car1.Start();//This is because we use Car()child extends Machine()parent

car1.wipedWindShield();

car1.showInfo();

car1.Stop();//So, Car() will inherit all the methods in Machines()

}

}

Class 1:

**public** **class** Machine {

**protected** String name = "I'm a machine";//This is a private var

//Using protected means you can access between anywhere in package & child class

**public** **void** Start(){

System.*out*.println("Machine Started. ");

}

**public** **void** Stop(){

System.*out*.println("Machine Stopped. ");

}

}

Class2:

**public** **class** Car **extends** Machine{

//extends--means Child is the child class of Machine()

/\*

public void Start(){ //Part I: Overwritting method

System.out.println("Car Started. ");//Overwritten method in parent class

//See that the method Start() has same name in Machine

//But Start in Car is executed. See output

}

\*/

//Part II: Another way to over-write method

//First comment Part I. Right Click in this editor. Go to Source

// Click Override... and in Machine() check the box Start()

//You should get the following

@Override

**public** **void** Start() {

//super.Start();//Part I :: Calling in parent class version

//you can add your own stuff in it

System.*out*.println("Car Started. ");//Part II

}

**public** **void** wipedWindShield(){

System.*out*.println("Wiping Wind Shield");

}

**public** **void** showInfo(){

System.*out*.println("Car name:" + name);

}

}

Console:

Machine Started.

Machine Stopped.

Car Started.

Wiping Wind Shield

Car name:I'm a machine

Machine Stopped.

Tutorial 23: Interfaces (watch video)

Main Class:

**public** **class** App {

**public** **static** **void** main(String[] args) {

Machine mach1 = **new** Machine();

mach1.start();

Person person1 = **new** Person("Bob");

person1.greet();

Info info1 = **new** Machine();

info1.showInfo();

Info info2 = person1;

info2.showInfo();

System.*out*.println();

*outputInfo*(mach1);

*outputInfo*(person1);

}

**private** **static** **void** outputInfo(Info info){

info.showInfo();

}

}

Class1:

**public** **class** Machine **implements** Info{

**private** **int** id = 7;

**public** **void** start(){

System.*out*.println("Machine STarted. ");

}

**public** **void** showInfo() {

System.*out*.println("Machine ID is : " + id);

}

}

Class2:

**public** **class** Person **implements** Info{

**private** String name;

//Right click -> Source -> Generate Constructor -> check "name"

**public** Person(String name) {

//super();//dont need

**this**.name = name;

}

**public** **void** greet(){

System.*out*.println("Hello There. ");

}

**public** **void** showInfo() {

System.*out*.println("Person name is : " + name);

}

}

Interface1: Use keyword implements in a class to implement “Interface”!!

//To create an interface -> Right Click on Tutorial->New->Interface->type name->Finish

**public** **interface** Info {

**public** **void** showInfo();//Interfaces dont have CURLY PARENTHESIS, No coding in it

}

Console:

Machine STarted.

Hello There.

Machine ID is : 7

Person name is : Bob

Machine ID is : 7

Person name is : Bob

Tutorial 24: Packages (Watch Video)

Main Code:

//Packages - 1) Enable you to organize your code

// 2) Stop you having conflicts between class name

//import ocean.Fish;//import from ocean package

//import ocean.Seaweed;//"Cntrl+Space"

//Or another way to import everything is use 'import ocean.\*'//means everything in ocean package!!

**import** ocean.Fish;//Cntrl+Shift+o

**import** ocean.Seaweed;

**public** **class** App {

**public** **static** **void** main(String[] args) {

Fish fish = **new** Fish();

Seaweed weed = **new** Seaweed();//type "Cntrl+Space" to automatically get "import" comand

}

}

Package 1 Class 1:

**package** ocean;//new package-should be first statement

**public** **class** Fish {

}

Package 1 Class 2:

**package** ocean;

**public** **class** Seaweed {

}

Package 1 Sub-Package 1 Class 1:

**package** ocean.plants;

**public** **class** Algae {

}

Console:

Tutorial 25: Public, Private and Protected

Main Code:

**import** world.Plant;

/\*

\* private -- only within same class

\* public -- from anywhere

\* protected -- same class, subclass, and same package

\* no modifier -- same package only (e.g int var1 -- no modifier, public int var1 -- has a modifier-public)

\*/

**public** **class** App {

**public** **static** **void** main(String[] args) {

Plant plant = **new** Plant();

System.*out*.println(plant.name);//out a class use plant.name, in the same class use this.name

//to access the variable

System.*out*.println(plant.*ID*);

//The following wont work because size is protected

//App is not in the same package as Plant

//System.out.println(plant.size);

}

}

**import** world.Plant;

**public** **class** Grass **extends** Plant{

**public** Grass(){

//Wont work -- grass in not same package as plant

//eventhough its in the same sub class

//System.out.println(this.height);

}

}

**package** world;

**public** **class** Plant {

//The following declaration is a bad practice

//You need to perform encapsulation

**public** String name;

//Acceptable practice -- its final.

**public** **final** **static** **int** *ID* = 7;

**private** String type;//Cannot acces outside this class

**protected** String size;

**int** height;

**public** Plant(){

**this**.name = "Freddie";//out a class use plant.name, in the same class use this.name

//to access the variable

**this**.type = "Plant";

**this**.size = "medium";

**this**.height = 8;

}

}

**package** world;

**public** **class** Oak **extends** Plant {

**public** Oak(){

//Following Wont work -- type is private

//type = "Tree";

**this**.size = "large";//This works because "protected" var size can be

//accessed within the class and subclass;

//Also within the same package

//this is use as accessor--Oak and Plant in same package

**this**.height = 10;

}

}

**package** world;//Plant is the world package

**public** **class** Field {

**private** Plant plant = **new** Plant();//create a new plant object

**public** Field(){

//size is protected;Field is in the same package as Plant()

System.*out*.println(plant.size);

}

}

Console:

Freddie

7

Tutorial 26: Polymorphism

**public** **class** App {

**public** **static** **void** main(String[] args) {

Plant plant1 = **new** Plant();

Tree tree = **new** Tree();

Plant plant2 = tree;//polymorphism in action

plant2.grow();//this method is from Tree not Plant

tree.shedLeaves();

*doGrow*(tree);

}

**public** **static** **void** doGrow(Plant plant){

plant.grow();

}

}

**public** **class** Plant {

**public** **void** grow(){

System.*out*.println("Plant is growing. ");

}

}

**public** **class** Tree **extends** Plant {

@Override

**public** **void** grow() {

System.*out*.println("Tree is Growing!!");

}

**public** **void** shedLeaves(){

System.*out*.println("Leaves are sheding");

}

}

Console:

Tree is Growing!!

Leaves are sheding

Tree is Growing!!

Tutorial 27: Encapsulation and API Docs

**class** Plant{

**public** **static** **final** **int** *ID* = 7;//final -- cant be changed

**private** String name;//Idea:: You encapsulate the inner working the class within the class

**public** String getData(){

String data = "some stuff " + calculateGrowthForecast();

**return** data;

}

**public** **int** calculateGrowthForecast(){

**return** 9;

}

//----- The following is created by

// Right Click in this editor -> Source -> Generate Getters and Setters

//Check the box for name

**public** String getName() {//Getters

**return** name;

}

**public** **void** setName(String name) {//Setters

**this**.name = name;

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

}

}

Console:

Tutorial 28: Casting Numerical Types

**public** **class** App {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**byte** byteValue = 20;

**short** shortValue = 55;

**int** intValue = 999;

**long** longValue = 234555;

**float** floatValue = 8843.6f;//use f at the end

//another way to create float variable

**float** floatValue2 = (**float**)99.23;

**double** doubleValue = 32.4;

System.*out*.println(Byte.*MAX\_VALUE*);// To check the max range

intValue = (**int**)longValue;

System.*out*.println(intValue);

doubleValue = intValue;//no cast before intValue

System.*out*.println(doubleValue);

intValue = (**int**)floatValue;

System.*out*.println(intValue);

//Following will give wrong value

//128 is too big for a byte

byteValue = (**byte**)128;

System.*out*.println(byteValue);

}

}

Console:

127

234555

234555.0

8843

-128

Tutorial 29: Up Casting and Down Casting

**class** Machine{

**public** **void** start(){

System.*out*.println("Machine Started. ");

}

}

**class** Camera **extends** Machine{

**public** **void** start(){

System.*out*.println("Camera Started!!! ");

}

**public** **void** snap(){

System.*out*.println("Photo taken. ");

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

Machine machine1 = **new** Machine();

Camera camera1 = **new** Camera();

machine1.start();

camera1.start();

camera1.snap();

//Up casting- you use it with variables. Once you created a variable

//polymorhism- object of child class

//Machine machine2 = new Camera();//This is polymorphism!!

//Above the relation=> Machine(dad)->Camera(son)->camera1(grandson)

Machine machine2 = camera1;//Up casting-up the family tree upto Machine

machine2.start();

//machine2.snap();//\*\*This wont work because its not in Machine class!!

//\*\*machine2 is Machine class type!!

//Down Casting

//The relation=> Machine(dad)->Camera(son)->machine3(g son)->camera2(gg son)

Machine machine3 = **new** Camera();

Camera camera2 = (Camera) machine3;//Very imp step. Java wants a confirmation from you

//that you know you are doing a downcast. Because downcasting is not safe. So put (Camera)

camera2.start();

camera2.snap();

}

}

Console:

Machine Started.

Camera Started!!!

Photo taken.

Camera Started!!!

Camera Started!!!

Photo taken.

Tutorial 30: Using Generics

**import** java.util.ArrayList;

**import** java.util.HashMap;

**class** Animal{

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

//////////// Before Java 5 /////////////

ArrayList list = **new** ArrayList();

list.add("apple");

list.add("banana");

list.add("orange");

String fruit = (String) list.get(1);//Down casting

System.*out*.println(fruit);

////////////Modern Java /////////////

ArrayList<String> strings = **new** ArrayList<String>();

strings.add("cat");//Adders

strings.add("dog");

strings.add("hen");

String animal = strings.get(1);//Getters

System.*out*.println(animal);

////////////More than one type arguments /////////////

HashMap<Integer,String> map = **new** HashMap<Integer,String>();

//////////// Java 7 /////////////

ArrayList<Animal> someList = **new** ArrayList<>();

}

}

Console:

banana

dog

Tutorial 31: Generics and Wildcards

**import** java.util.ArrayList;

// Advanced Topic //

**class** Machine {

//Right click->Source->Generate toString()...->drop down ->Check toString()

@Override

**public** String toString() {

**return** "I am a Machine";

}

**public** **void** start(){

System.*out*.println("Machine Starting. ");

}

}

**class** Camera **extends** Machine{

@Override

**public** String toString() {

**return** "I am a Camera";

}

**public** **void** snap(){

System.*out*.println("Snap. ");

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

/\* PART I: How to pass parameter of ArrayList to a method //

ArrayList<String> list = new ArrayList<String>();//Now passing list as a parameter!!

list.add("one");

list.add("two");

showList(list);//to a new method called showList()!!

\*/

//Part II now using Machine as ArrayList!! String is predefined Method()

ArrayList<Machine> list1 = **new** ArrayList<Machine>();//compare above Machine() instead of String()

list1.add(**new** Machine());//compare above Machine() instead of String()

list1.add(**new** Machine());//compare above Machine() instead of String()

//---"ArrayList" of child class Camera is not same as "ArrayList" of parent Machine---//

//--- So use a wildcard indicated by "?", when you dont know the class type!!

ArrayList<Camera> list2 = **new** ArrayList<Camera>();//compare above Machine() instead of String()

list2.add(**new** Camera());//compare above Machine() instead of String()

list2.add(**new** Camera());//compare above Machine() instead of String()

*showList*(list1);

*showList*(list2);//After using wild card

}

/\* PART I:

public static void showList(ArrayList<Machine> list){//compare above Machine() instead of String()

for(Machine value: list){//compare above Machine() instead of String()

System.out.println(value);

}

}

\*/

/\* PART II:

public static void showList(ArrayList<?> list){// "?" wildcard: You can pass any type of parameter

// Con:You treat every para as "Object". You cant call methods, but only thro'

//ArrayList<MethodName>()

for(Object value: list){//use Object - because you dont know the type

System.out.println(value);

}

}\*/

//Part III: Upper Bound avoiding Object technique

**public** **static** **void** showList(ArrayList<? **extends** Machine> list){

**for**(Machine value: list){//No error without use of Object

System.*out*.println(value);

value.start();

}

}

}

Console:

I am a Machine

Machine Starting.

I am a Camera

Machine Starting.

I am a Camera

Machine Starting.

Tutorial 32: Anonymous Classes (Watch Video)

**class** Machine{

**public** **void** start(){

System.*out*.println("Start A Machine...");

}

}

**interface** Plant{

**public** **void** grow();//interface no "{}"

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

Machine machine1 = **new** Machine(){

@Override **public** **void** start(){

System.*out*.println("Camera Snapping...");

}

};

machine1.start();

Plant plant1 = **new** Plant(){

@Override

**public** **void** grow() {

// **TODO** Auto-generated method stub

System.*out*.println("Plant Growing...");

}

};

plant1.grow();

}

}

Console:

Camera Snapping...

Plant Growing...

Tutorial 33: Reading Text Files

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.Scanner;

**public** **class** App {

**public** **static** **void** main(String[] args) **throws** FileNotFoundException {

String fileName = "C:/Users/student/Desktop/example.txt";//Use Forward slashes / / /

File textFile = **new** File(fileName);//Memory for file

Scanner in = **new** Scanner(textFile);//Read File

**int** value = in.nextInt();//Find the first integer value!!

System.*out*.println("Read value: "+value);

in.nextLine();//to avoid reading '\n' character

**int** count = 2;

**while**(in.hasNextLine()){

//As long there are next line keep reading

String line = in.nextLine();//Read the text

System.*out*.println(count + ":" + line);

count++;

}

in.close();//make sure to close file

}

}

Console:

Read value: 3

2:line one

3:line two

4:line three

5:

6:cat

7:fox

8:dog

9:cat

Tutorial 34: Handling Exceptions

Console:

Tutorial 35:

Console:

Tutorial 36:

Console:

Tutorial 37:

Console:

Tutorial 38:

Console:

Tutorial 39:

Console:

Tutorial 39:

Console:

Tutorial 40:

Console:

Tutorial 41:

Console:

Tutorial 42:

Console:

Tutorial 43:

Console:

Tutorial 44:

Console:

Tutorial 45:

Console:

Tutorial 46:

Console:

Tutorial 47:

Console: