|  |  |  |
| --- | --- | --- |
| **Concepts/Code** | **Links/Notes** | **Duration** |
| Installing Java | [**https://www.java.com/en/download/help/download\_options.xml**](https://www.java.com/en/download/help/download_options.xml) |  |
| Installing Eclipse | [**https://www.eclipse.org/downloads/eclipse-packages/?show\_instructions=TRUE**](https://www.eclipse.org/downloads/eclipse-packages/?show_instructions=TRUE) |  |
| how to familiar with eclipse IDE | [**https://www.tutorialspoint.com/eclipse/eclipse\_create\_java\_class.htm**](https://www.tutorialspoint.com/eclipse/eclipse_create_java_class.htm) |  |
| Creating Class:  Ex:  public class DataArtist {  } | Open Eclipse then  create new project and package and then add new class with .java extension**.** |  |
| Creating Variables: | Follow this link and write code:  <http://www.learnjavaonline.org/en/Variables_and_Types> |  |
| creating method with void and no parameter  public void draw() //signature  {  //method body  }  Ex: Modify DataArtist with draw method in Eclipse  public class DataArtist {  public void draw() {    }   } | we write methods in class. We write logic(our test case steps) in side method. it will contain two parts:  a.*method signature*  method signature will contain:   1. Modifiers—such as public, private, and others you will learn about later. 2. The return type—the data type of the value returned by the method, or void if the method does not return a value. 3. The method name— 4. The parameter list in parenthesis—If there are no parameters, you must use empty parentheses.   B. The method body, enclosed between braces—the method's code, including the declaration of local variables, goes here. |  |
| creating method with void and parameters  Ex: Add two new methods, draw1, AddShape to DataArtist class like below:  public class DataArtist {  public void draw() { //void and no parameters    }  public void draw1(int i) { // void and parameters    }  public void AddShape(int i, String y) {    }  } | We declare parameters with in parentheses of method signature. We declare parameters  just like variables. Parameters scope is with in method body only not outside of method.  we specify multiple parameters with comma delimiter. |  |
| creating method with return type and no parameter  Ex: Add two new methods, drawShape, drawShape2 to DataArtist class like below:  public class DataArtist {  public void draw() { //void and no parameters    }  public void draw1(int i) { // void and parameters    }  public void AddShape(int i, String y) {    }  //this one returns directly value  public int drawShape() {  return 2;  }  //use variable after return keyword  public int drawShape2() {  int result;  return result;  }  } | in place of **void** we specify data type (ex: int, double, String) in method signature.  we need to use **return** keyword inside method.  After **return** keyword we can return directly value (like 2) or variable of type matching with return type specified before method name. |  |
| creating method with return type and parameter:  public class DataArtist {  public int drawShape() {  return 2;  }  //use variable after return keyword  public int drawShape2() {  int result;  return result;  }  public int drawShape3(int x) {  return 2;  }  //use variable after return keyword  public int drawShape4(int x, String y) {  int result;  result = x \* 2;  return result;  }  } | we specify parameters in parentheses after method name. In place of void we specify data type (ex: int, clasname, double) in method signature  we need to use return keyword inside method. After return keyword we can return directly value (like 2 or variable of type matching with return type specified before method name). |  |
| purpose of main method  Ex:  Public class DataArtist{  Public static main(String[] args){  }  } | A Java program starts by executing the main method of some class. You can choose the name of the class to execute, but not the name of the method. The method must always be called main. |  |
| declaring properties/data members/fields with different data types.  Ex:  Public class DataArtist{  public int cadence;  public String gear;   public float speed;  Public static main(String[] args){  }  } | we declare properties/fields same way we declare variables (in above example we declared variables inside methods) but we declare properties/fields under class (outside of methods).  we can access these properties/fields in methods we declare in class.  Data members/properties. |  |
| creating object from class:  Ex: className objectname = new ClassName();  Public class DataArtist{  public int cadence;  public String gear;  public float speed;  Public static main(String[] args){  DataArtist data = new DataArtist();  //here "data" is object name.  }  } | whenever we wanted to use class properties/methods we need to create object.  we create object with "new". |  |
| assigning value to property:  Public class DataArtist{  public int cadence;  public String gear;  public float speed;  Public static main(String[] args){  DataArtist data = new DataArtist();  data.cadence = 20;  data.gear = "Second";  //we use double quotes when we assign string data to field.  }  } | as we discussed properties will hold data of class. to assign value to above properties in main method write below things: |  |
| reading value from property  Ex:  Public class DataArtist{  public int cadence;  public String gear;  public float speed;  Public static main(String[] args){  DataArtist data = new DataArtist();  data.cadence = 20;  data.gear = "Second";  int x = data.cadence;  //we are reading "cadence" property value and store into variable "x"  String test = data.gear;  //we are reading "gear" property value and store into variable "test".  }  } | we can read value from property and store into variable |  |
| **calling method with void and no parameter**  **Ex:**  Public class DataArtist{  public void draw() {   }  Public static main(String[] args){  DataArtist data = new DataArtist();  data.draw();  //we are calling draw method which does not take any parameters  }  } | whenever we wanted to use method (we created several methods in above steps)  we need to call method with objectname.methodname(); |  |
| **calling method with void and parameter**  Public class DataArtist{  public void draw() {     }  public void draw1(int i) {    }  public void AddShape(int i, String y) {    }  Public static main(String[] args){  DataArtist data = new DataArtist();  data.cadence = 20;  data.gear = "Second";  int x = data.cadence;  String test = data.gear;  data.draw1(x);  data.AddShape(x, test);  }  } | whenever we wanted to use method (we created several methods in above steps)  we need to call method with objectname.methodname(pram1, param2);  we declared x, test as variables and it has values from properties cadence  and gear. |  |
| **calling method with return type and no parameter**  Public class DataArtist{  public void draw() {     }  public void draw1(int i) {    }  public void AddShape(int i, String y) {    }  public int drawShape() {  return 2;  }  public int drawShape2() {  int result;  return result;  }  public int drawShape3(int x) {  return 2;  }  public int drawShape4(int x, String y) {  int result;  result = x \* 2;  return result;  }  Public static main(String[] args){  DataArtist data = new DataArtist();  data.cadence = 20;  data.gear = "Second";  int x = data.cadence;  String test = data.gear;  data.draw1(x);  data.AddShape(x, test);  int y = data.drawShape();  }  } | when ever we wanted to use method (we created several methods in above steps)  we need to call method with objectname.methodname();  //we are calling drawShape method with no parameter. |  |
| **calling method with return type and parameter**  Public class DataArtist{  public void draw() {     }  public void draw1(int i) {    }  public void AddShape(int i, String y) {    }  public int drawShape() {  return 2;  }  public int drawShape2() {  int result;  return result;  }  public int drawShape3(int x) {  return 2;  }  public int drawShape4(int x, String y) {  int result;  result = x \* 2;  return result;  }  Public static main(String[] args){  DataArtist data = new DataArtist();  data.cadence = 20;  data.gear = "Second";  int x = data.cadence;  String test = data.gear;  data.draw1(x);  data.AddShape(x, test);  int y = data.drawShape();  int z = data.drawShape3(x);  int w = data.drawShape4(x, test);  }  } | when ever we wanted to use method (we created several methods in above steps)  we need to call method with objectname.methodname(pram1, param2);  //we are calling drawShape3 method with parameter "x".  //we are calling drawShape4 method with parameters "x" and "test". |  |
| **store return value from method in variable:**  write below code in main method:  int shape3 = data.drawShape3(x);  write below code in main method:  int shape4 = data.drawShape4(x, test); | when ever we wanted to use method (we created several methods in above steps)  we need to call method with objectname.methodname();  some with methods return value (we learned above by adding return keyword inside method body)  //we are calling drawShape3 method with parameter "x"  in above code, we declared x as variable and it has value from property "cadence".  what ever value is coming from drawShape3 and we are storing that value in "shape3" variable.  here datatype of variable and datatype we specified in method declaration (before method name in place of void) should be same  //we are calling drawShape4 method with parameters "x" and "test"  in above code, we declared x, test as variables and it has values from properties cadence and gear. |  |
| **Inheritance:creating child class from parent class**  **Ex:**  **Parent Class:**  public class Personal  {  private String name,gender;  private int age;  public Personal(String n,String g,int a){  name=n;  gender=g;  age=a;  }  public String getName(){  return name;  }  public void setName(String name){  this.name = name;  }  **}**  **Child Class**  Here Academic class created as sub class/derived class/child class  public class Academic extends Personal //super class (Personal),Derived class(Academic)  {  private String standard,university;    public Academic(String a,String u,String name,String g,int ag){  super(name,g,ag);  standard=a;  university=u;  }  public String getStandard() {  return standard;  }  public String getUniversity() {  return university;  }    public static void main(String args[]) {  Academic a=new Academic("B.tech","JNTU","rakesh","male",21);  System.out.println(a.getName());  System.out.println(a.getStandard());  System.out.println(a.getUniversity());  }  } | When we want to use a class properties in other class then we must use a feature inheritance. In the example given below personal class created as super class and academic class created as sub class. To inherit a class in other a keywork should be used i.e extends.  // we are using parent property here. |  |
| **override: parent method in child class**  **Ex:**  **Parent Class**  public class Shape  {  double dim1,dim2;  public Shape(double d1,double d2){  dim1=d1;  dim2=d2;  }  public double Area(){  return 0.0;  }  }  **Child Class:**  public class Rectangle extends Shape  {  public Rectangle(double d1,double d2){  super(d1,d2);  }  public double Area(){  //**overriding super class Method**  return dim1\*dim2;  }  public static void main(String args[]) {  Rectangle re=new Rectangle(10.0,20.0);  System.out.println("Area:"+re.Area());  }  } | When a super class method re implemented or redefined in subclass , then we can call that as overriding. As per the application requirements we can do that. In the example given below Area() method overridded in sub class/Child class/Derived class **Rectangle**. Here super class/base class/base class is **Shape**. |  |
| **Constructor**  public class Box {  Public double length,breadth,height;  public Box(){ //**constructor**  length=breadth=height=5;  }  public Box(double l,double b,double h){  length=l;  breadth=b;  height=h;  }    public static void main(String args[]) {  Box b=new Box();  //Box b1=new Box(10,20,20);    System.out.println(b.volume());  //System.out.println(b1.volume());  }  } | Any Method created with class name and doesn’t have any return type we can call that as constructor. constructor executes after object created. It allows us to initialize the object.  //constructor |  |
| **create interface with 2 methods with declarations**  **Example**:  Public interface MyInterface  {  Public void accept();  Public void Show();  } | What are the methods created in a interface, creates as abstract methods. Any variable created in the interface, those are comes as static final. It allows us to do abstraction and multiple inheritance. |  |
| **create class that implements interface:**  Public class MyClass implements MyInterface{  Public void accept(){  System.out.println(“This Method from interface”);  }  Public void Show(){  System.out.println(“This Method from interface”);  }  } | When we want to implement the methods created in the interface , then we must create class and those methods should be implemented. While implementing those methods we must use a a keyword **implements** while creating class. |  |
| **learn about how to create static methods/properties**  **Example:**  public class StaticBank {  String accountholdername;  double balance;  int accountnumber;  static String bankname="SBI"; //**static variable**    public String getAccountholdername()  {  return accountholdername;  }  public void setAccountholdername(String accountholdername)  {  this.accountholdername = accountholdername;  }  public int getAccountnumber()  {  return accountnumber;  }  public void setAccountnumber(int accountnumber)  {  this.accountnumber = accountnumber;  }  public double getBalance()  {  return balance;  }  public void setBalance(double balance)  {  this.balance = balance;  }  public static double FindInterest(StaticBank b){  b.balance=(b.balance+b.balance\*0.04);  return b.balance;  }  public static void main(String args[]){  StaticBank sb1=new StaticBank();  sb1.setAccountholdername("Rakesh");  sb1.setAccountnumber(1002);  sb1.setBalance(4200);    StaticBank sb2=new StaticBank();  sb2.setAccountholdername("Kishore");  sb2.setAccountnumber(1004);  sb2.setBalance(8200);  double result1=FindInterest(sb1);  System.out.println("After Interest calculation:"+result1);  double result2=FindInterest(sb2);  System.out.println("After Interest calculation:"+result2);  }  } | Static Methods can be called with class name i.e without object. When we want to use a method on multiple objects then we must create that method as static method. In the same way if we want to use common value for multiple objects then that value should be create with static variable . all static methods can be called with class name not with the object.  //static method  // static method call  //static method call |  |
| **handling exceptions with try/catch/block**  public class Example {  public static void main(String args[]){  try{  int a=10;  int count=args.length;  if(count==0)  throw new ArithmeticException("please check denominator.....");  int result=a/count; //arithematic System.out.println("Result :"+result);  }  catch(ArithmeticException ar) {  System.out.println(ar.getMessage());  //System.out.println("Denominator is zero"); }  catch(ArrayIndexOutOfBoundsException ai){  //System.out.println("invalid index");  }  finally{  }  }  } | **Exception**  when any programme execution stopped suddenly due to some problem, then it is known as abnormal termination. So when any programme terminated abnormally, to handle that we can use technique called exception handling. In java language to handle exceptions we should use try...catch blocks . |  |
| **learn about public, private, default, protected access modifiers**  public class Shape  {  protected double dim1,dim2;    public Shape(double d1,double d2){  dim1=d1;  dim2=d2;  }  public double Area(){  return 0.0;  }  }  public class Rectangle extends Shape {  public Rectangle(double d1,double d2){  super(d1,d2);  }  public double Area(){ //overriding  return dim1\*dim2; //variables created as protected in super class i.e shape class  }  public static void main(String args[]){  Rectangle re=new Rectangle(10.0,20.0);  System.out.println("Area:"+re.Area());  }  } | Private: all private properties in a class are accessible only in the declared class.  Public: all public properties in a class are accessible only in the declared class as well outside of the class also.  Protected: all protected properties in a class are accessible in the declared class as well derived class also. |  |
| **creating packages**  Example:  package employeePack; //package declaration  import java.io.Serializable;  public class Employee implements Serializable{  private int id;  private String name;    public Employee(){    }  public void setId(int id){  this.id=id;  }  public int getId(){  return id;  }  public void setName(String name){  this.name=name;  }  public String getName(){  return name;  }  } | **Package** in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. When we want to create set of classes and interfaces then althose should be created under a package. |  |
| **import packages and uses classes from packages**  package letmecalculate;  public class Calculator {  public int add(int a, int b){  return a+b;  }  public static void main(String args[]){  Calculator obj = new Calculator();  System.out.println(obj.add(10, 20));  }  }  ---------------------------------------------------  import letmecalculate.Calculator; //importing classes  public class Demo{  public static void main(String args[]){  Calculator obj = new Calculator();  System.out.println(obj.add(100, 200));  }  } | To access a class which is already created in a package we must import that package by using a keyword **import** |  |
| **IF STATEMENT**  Example:  public class IfStatement{  int num=10;    public void CreateIfStatement() {  if( num < 100 ) {  System.out.println("number is less than 100");  }  }  public static void main(String args[]) {  IfStatement i=new IfStatement();  i.CreateIfStatement();  }  } | The statements gets executed only when the given condition is true. If the condition is false then the statements inside if statement body are completely ignored.  /\* This println statement will only execute, \* if the above condition is true \*/ |  |
| **Switch**  Example:  public class SwitchCaseExample{  public static void main(String args[]) {  int num=0;  switch(num) {  case 1:  System.out.println("Given Value is 1");  break;  case 2:  System.out.println("Given Value is 2");  break;  default:  System.out.println("Given Value is 0");  }  }  } | Switch case statement is used when we have number of options (or choices) and we may need to perform a different task for each choice. |  |
| **While loop**  Example:  class WhileLoopExample {  int i=10;  public void WhileLoop() {  while(i>1){  System.out.println(i);  i--;  }  }  public static void main(String args[]) {  WhileLoopExample wle=new WhileLoopExample();  wle.WhileLoop();  }  } | In while loop, condition is evaluated first and if it returns true then the statements inside while loop execute. When condition returns false, the control comes out of loop and jumps to the next statement after while loop.While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop.** |  |
| **For Loop**  Example:  class ForLoopExample {  int i;  public void CreateForLoop() {  for(int i=1; i<=10; i++) {  System.out.println("The value of i is: "+i);  }  }  public static void main(String args[]) {  ForLoopExample fle=new ForLoopExample();  fle.CreateForLoop();  }  } | Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true. For loop contains 3 parts initializaion(first part),  condition (second part),  expression(third part). |  |
| **Foreach Loop**  **Example**:  class ForEachExample{  int arr[]={12,13,14,44}; // Array Declaration  int i;  public void CreateForEachLoop() {  for(int i:arr) {  System.out.println(i);  }  }  public static void main(String args[]) {  ForEachExample fe=new ForEachExample();  fe.CreateForEachLoop();  }  } | While using foreach loop, to display the values we can use reduce the code and there is no use of the index or rather the counter in the loop. So that it is easy to use.For-Each Loop is another form of for loop used to traverse the array. |  |
| **Do-while Loop**:  Example:  class DoWhileLoopExample2 {  int arr[]={2,11,45,9}; // array declaration  //i starts with 0 as array index starts with 0  int i=0;  public void ShowValues() {  do {  System.out.println(arr[i]);  i++;  }while(i<4);  }  public static void main(String args[]) {  DoWhileLoopExample2 dwe=new DoWhileLoopExample2();  dwe.Showvalues();  }  } | First, the statements inside loop execute and then the condition gets evaluated, if the condition returns true then the control gets transferred to the “do” else it jumps to the next statement after do-while. |  |
| **Continue:**  **Example:**  public class ContinueExample{  public static void main(String args[]){  for (int j=0; j<=6; j++){  if (j==4) {  continue;  }  System.out.print(j+" ");  }  }  } | Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running the loop but stop processing the remainder of the code in its body for this particular iteration. This is, in effect, a goto just past the body of the loop, to the loop’s end. The continue statement performs such an action. |  |
| **Break:**  **Example:**  public class BreakExample{  int num =0;  public void Display() {  while(num<=100) {  System.out.println("Value of variable is: "+num);  if (num==2) {  break;  }  num++;  }  System.out.println("Out of while-loop");  }  public static void main(String args[]){  BreakExample be=new BreakExample();  be.Display();  }  } | Using break, we can terminate of a loop, by giving the condition. So when the given condition is true then programme execution stops and comes out of the loop. |  |

|  |  |
| --- | --- |
| Overload vs overriding  Compile time poly vs Run Time Poly |  |
| Throw vs throws |  |
|  |  |

**Collections:**

|  |  |
| --- | --- |
| **ArrayList:**  **Creating :**  Syntax:  List objName = new ArrayList<objectType>();  Ex:  List fruits = new ArrayList<String>();  **Add item:**  fruits.add(“apple”);  fruits.add(“banana”);  fruits.add(“orange”);  fruits.add(3,”pineapple”);  fruits.add(4,”berry”);  fruits.add(“apple”);  **Read Item:**  fruits.get(0); //apple  fruits.get(2); // orange  for(String item : fruits){  System.out.println(item);  }  **Update Item:**  fruits.set(2,”strawberry”);  **Delete Item:**  fruits.remove(0) //apple is removed  fruits.remove(new String(“orange”));  //orange is removed. | Arraylist implements List interface, it allows duplicates items,  //add at end of lists  //add at specified index  // retrieve a particular item at index  //retrieve all the items in list  // to replace item at index 2  //replace banana with strawberry.  //to remove at particular index  //to remove a particular item if we don’t know index. |
| **HashSet:**  **Creating:**  HashSet<String> num = new HashSet<String>();  **Add Item:**  num.add(“one”);  num.add(“two”);  num.add(“three”);  **Read Item:**  Iterator<String> i = num.iterator();  while(i.hasNext()){  System.out.println(i.next());  }  System.out.println(num);  **Delete item:**  num.remove(“three”); | HashSet implements set interface, doesn’t allow duplicate items. |
| **TreeSet:**  **Creating:**  TreeSet<String> alp = new TreeSet<String>();  **Add Item:**  alp.add(“A”);  alp.add(“B”);  alp.add(“C”);  **Read Item:**  Iterator<String> i = alp.iterator();  while(i.hasNext()){  System.out.println(i.next());  }  System.out.println(alp);  **Delete item:**  alp.remove(“B”); | TreeSet implements set interface, doesn’t allow duplicate items. It stores items in sorted order. |
| **HashMap:**  **Creating:**  HashMap<String, Integer> marks= new HashMap<>();  **Add Item:**  marks.put(“java”, 80);  marks.put(“selenium”,90);  marks.put(“appium”,85);  **Read Item:**  marks.get(“java”);  **Update Item:**  marks.put(“java”,88); | Hashmap implements map interface, stores key value pairs, keys are unique and allows one null as key, |
| **TreeMap:**  **Creating:**  TreeMap<Integer, String> nums = new TreeMap<>();  **Add Item:**  nums.put(2,”two”);  nums.put(3,”three”);  nums.put(1,”one”);  **Read Item:**  map.get(1);  **Update Item:**  map.put(1,”oneee”); | Treemap implements map interface, stores key value pairs, keys are unique. Store in sorted order of keys. |