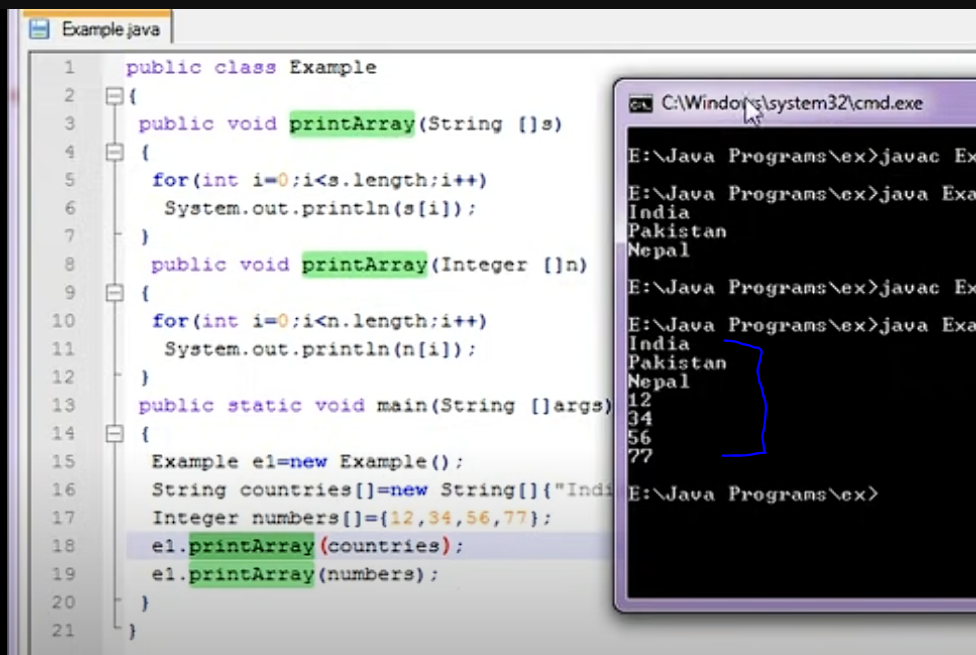
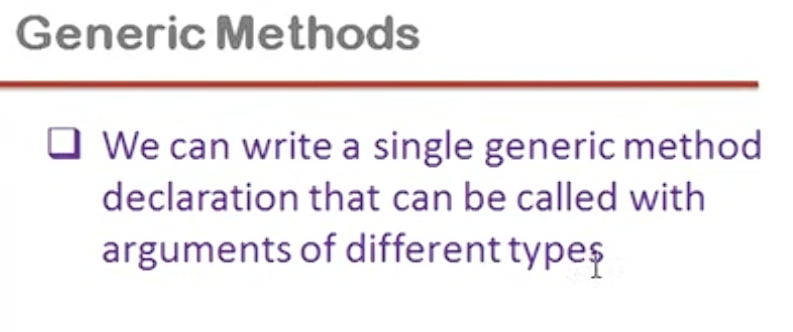


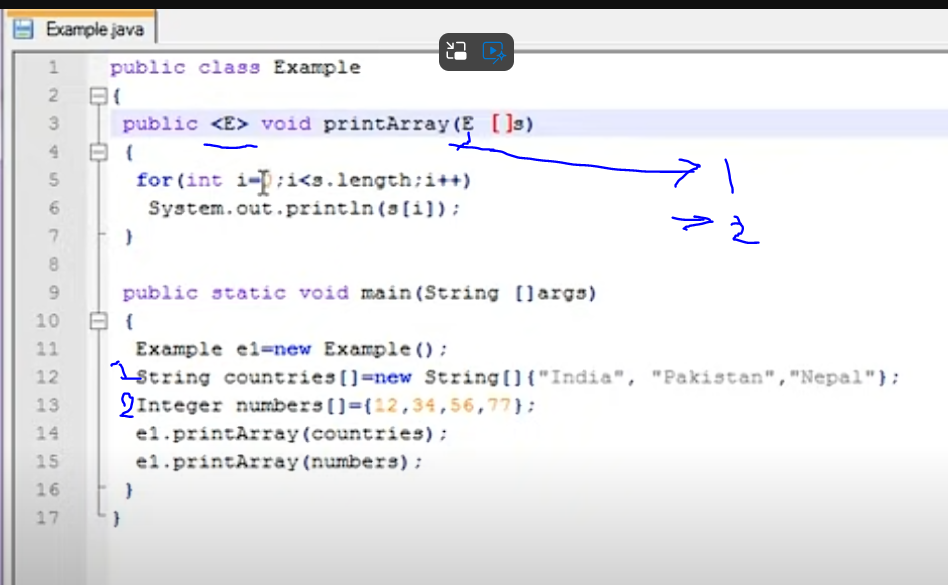
A screenshot of a computer program

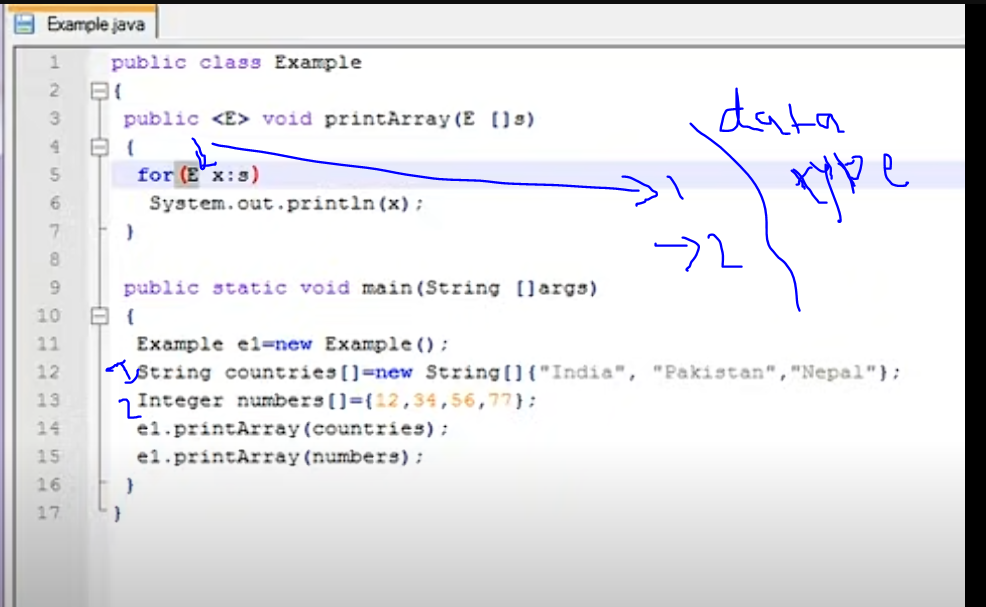
Description automatically generated

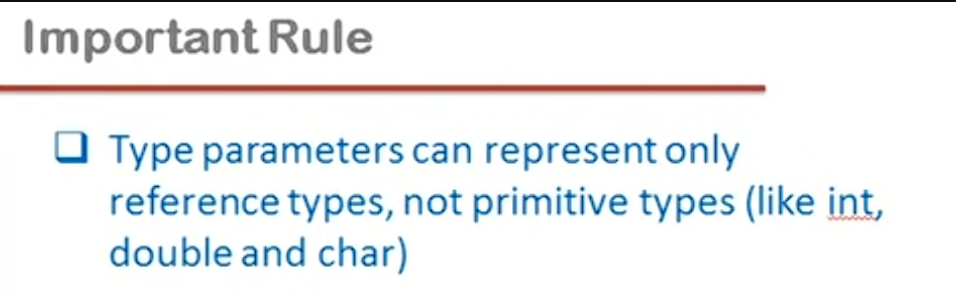


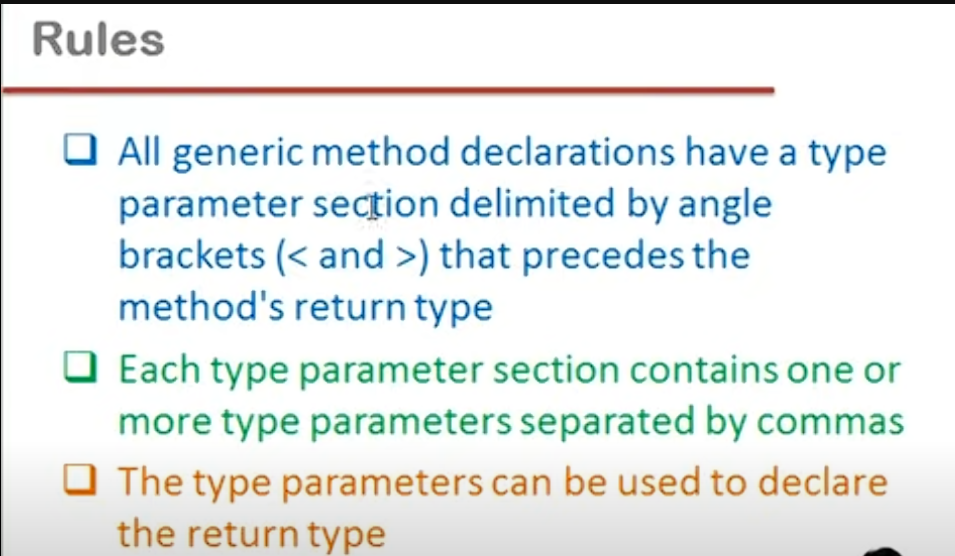
Generic Function : data type of arguments will become/converted as data type of incoming/receiving arguments.

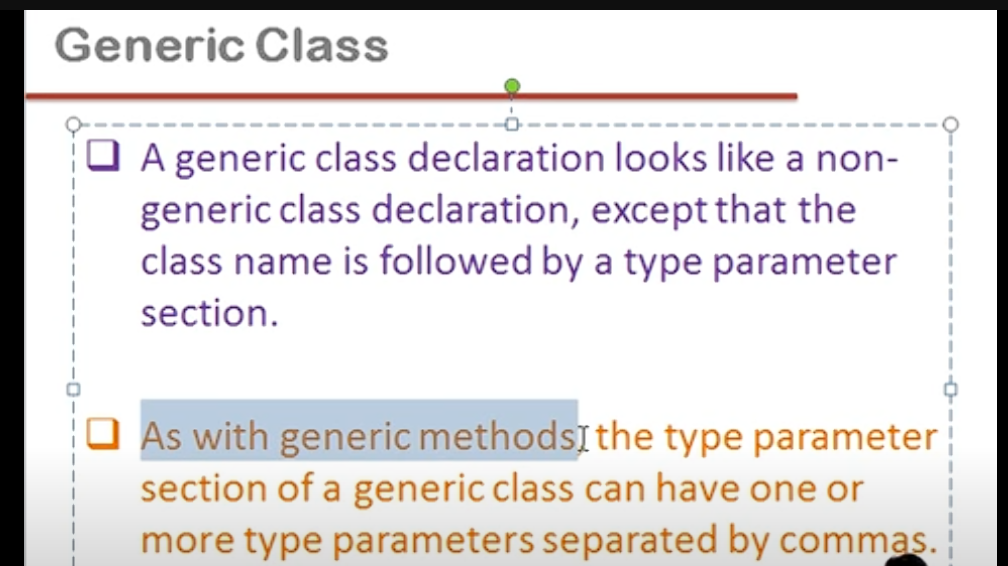


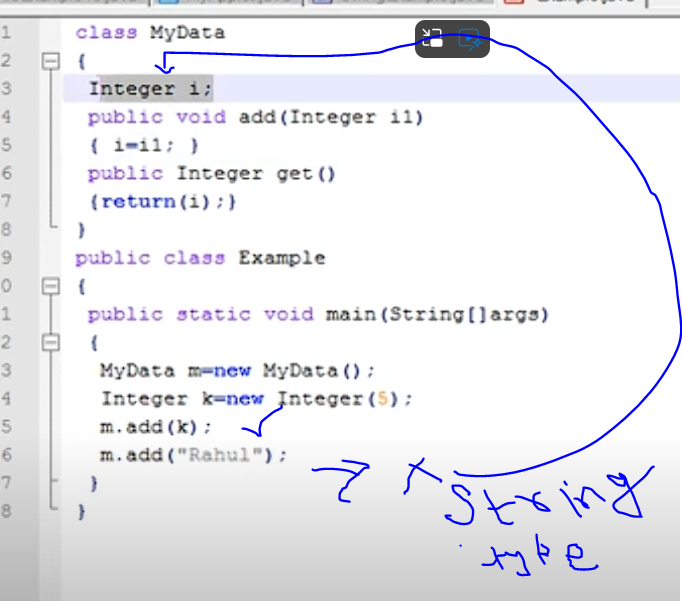










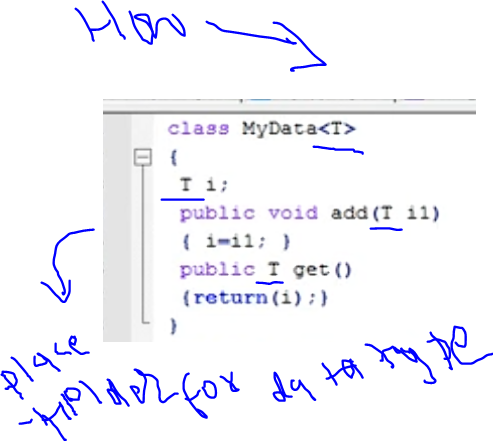


A drawing of a diagram

Description automatically generated

So in-order to keep various different types of data in MyData Object , we will have to create different types of classes with required type of Data type in each one ,so foe example , to keep float value in MyData object , will have to create a class with float type of instance variable and so on…

So , in order to generalize the type for object of a class , we need Generic class.



How the Type of T will be determined: at the time of Object Creation

A computer screen shot of a computer code

Description automatically generated

Bound In Generics

:

Only extended class Object can be passed , not others, also we can extend other Interfaces as well. We can call methods of Parent class as well.

public class Animal {  
  
 int legs;  
 float weight;  
  
 public Animal(int legs, float weight) {  
 this.legs = legs;  
 this.weight = weight;  
 }  
  
 public void eat()  
 {  
 System.*out*.println("i am annimal eat");  
 }  
}

public class Cat extends Animal{  
String speak;  
  
 public Cat(int legs, float weight,String speak) {  
 super(legs, weight);  
 this.speak = speak;  
 }  
}

public class Dog extends Animal{  
 String speak;  
  
 public Dog(int legs, float weight,String speak) {  
 super(legs, weight);  
 this.speak = speak;  
 }  
}

import java.io.Serializable;  
  
public class Printer<T extends Animal > {  
 T thingsToPrint;  
 public Printer(T thingsToPrint)  
 {  
 this.thingsToPrint = thingsToPrint;  
 }  
 public void printThings()  
 {  
 thingsToPrint.eat();  
 System.*out*.println(thingsToPrint);  
 }  
}

public class GenericExample  
{  
 public static void main(String[] args)  
 {  
 System.*out*.println("Generic Example");

// class Printer<T> -- in this only below will work  
*// Printer<Integer> printer = new Printer<>(23);  
// printer.printThings();  
// Printer<Double> dprinter = new Printer<>(23.4);  
// dprinter.printThings();*

*// with Bound Generics* Printer<Cat> printer = new Printer<>(new Cat(4,20.4f,"meow"));  
 printer.printThings();  
 Printer<Dog> dprinter = new Printer<>(new Dog(4,40.2f,"Bark bow bow"));  
 dprinter.printThings();  
 }  
}

Method :

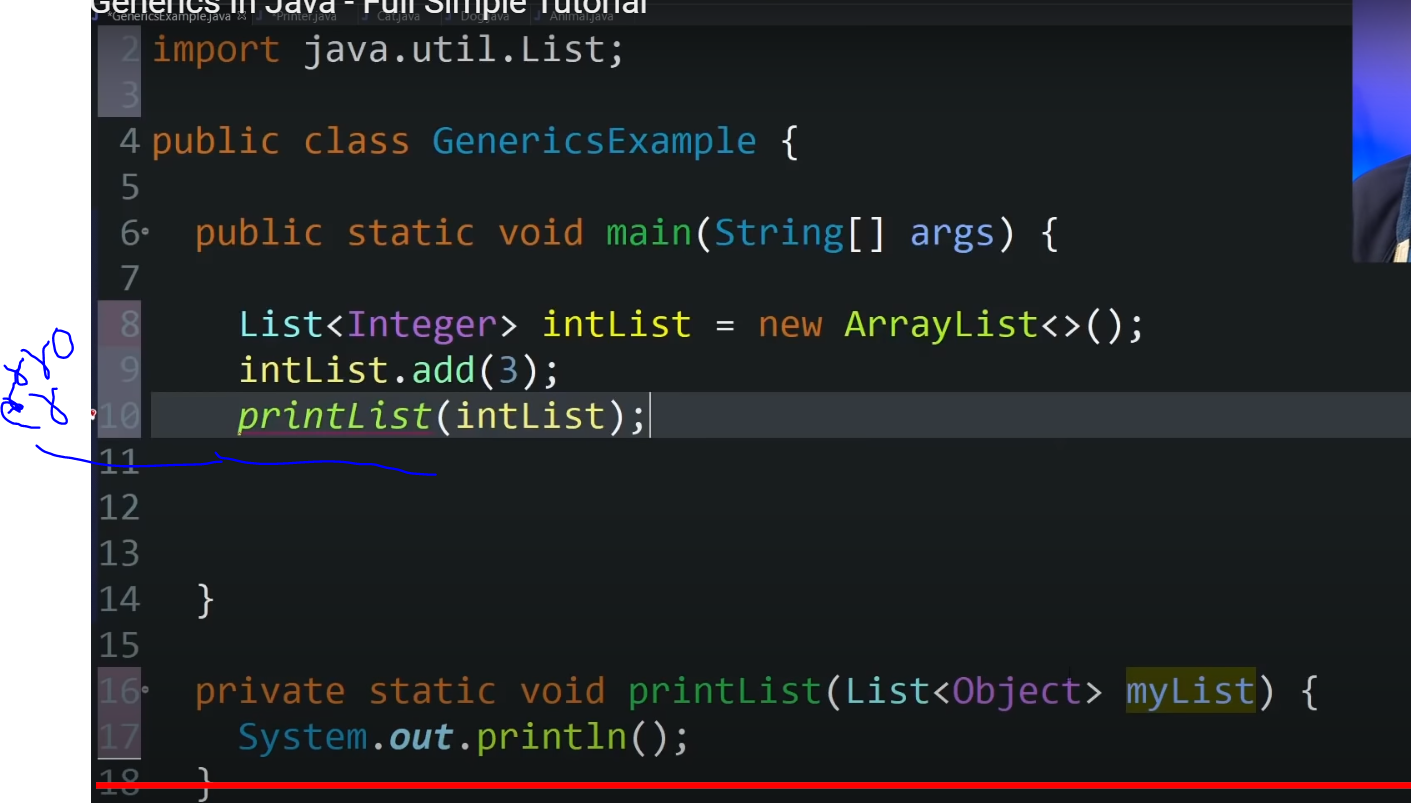
public class GenericExample  
{  
 public static void main(String[] args)  
 {

// with single Param *// shout("acd");  
// shout(123);*

*//with more than one param*

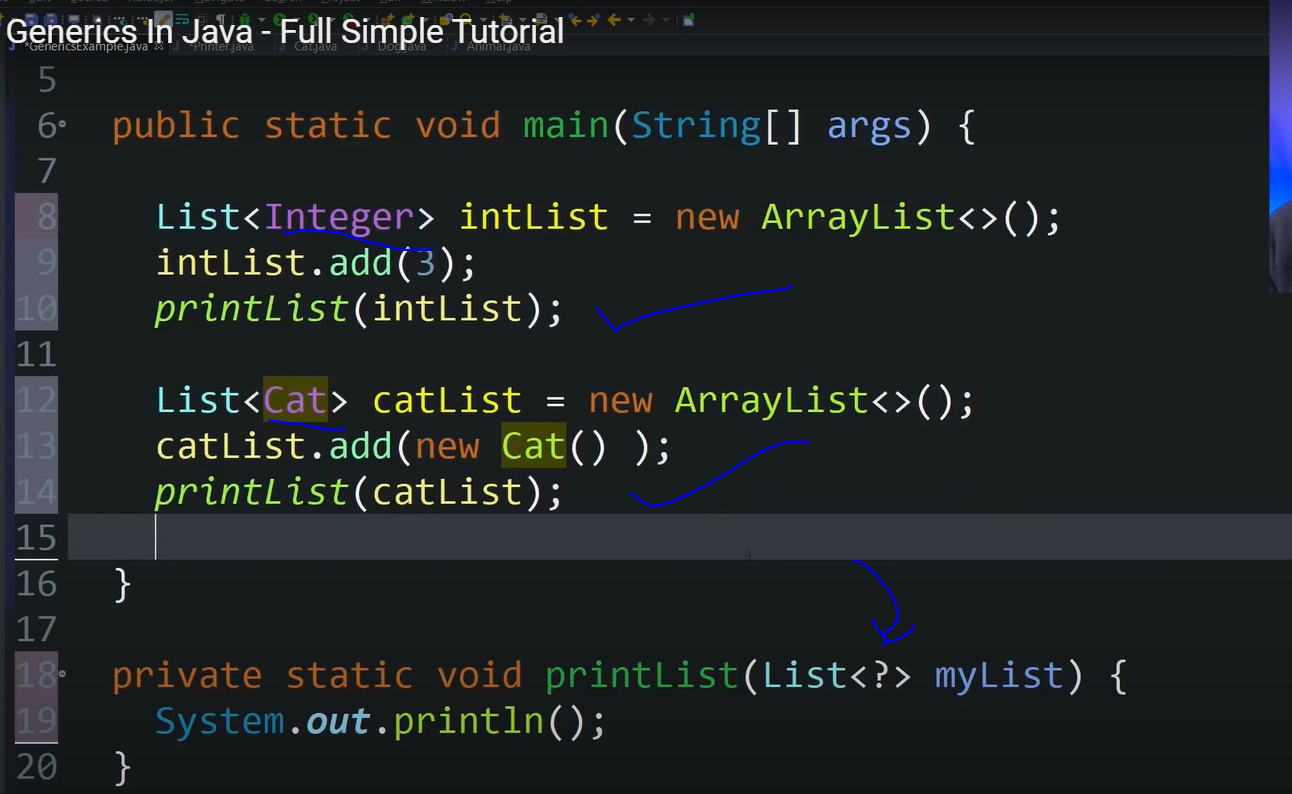
*shout*("acd",1234);  
 *shout*(123,34.8);  
  
  
 }  
 private static <T,V> T shout(T thingstoshout , V ootherthingstoshout)  
 {  
 System.*out*.println(thingstoshout +"!!!!!!");  
 System.*out*.println(ootherthingstoshout +"!!!!!!");  
 return thingstoshout;  
 }  
}-----

Wildcards



List Of integer is not subclass of List of Objects , so above error. So complier gives error , though Integer is subclass of Object , not the List.

Solution to above problem : wildcard , i.e ?, List of Unknown .



Bound with Wildcards,

