Java Basics

**Getter and Setter practical example**

When you are accelerating your car cars accelerator pedal in your game, you do car.setSpeed(increased\_value);   
To read the current speed (for example, to display it on screen or calculate some physics), you do car.getSpeed() and use that value as you want to

Eg :

void setSpeed (String t) {

               Speed = t;

            }

 Int Speed = 65;

String getSpeed() {

            return Speed;

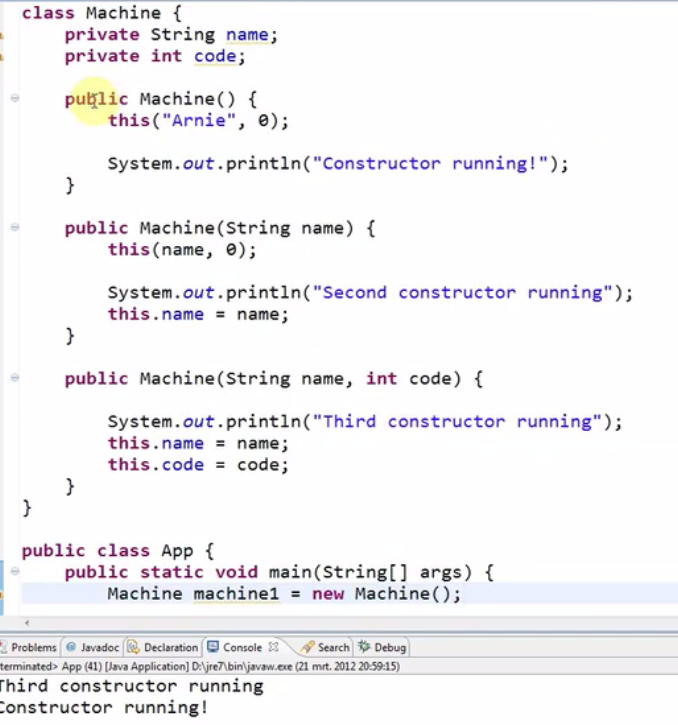
     }

Constructor

* A constructor doesn’t have a return type.
* The name of the constructor must be the same as the name of the class.
* Unlike methods, constructors are not considered members of a class.
* A constructor is called automatically when a new instance of an object is created.

Like methods, constructors can be overloaded. In other words, you can provide more than one constructor for a class if each constructor has a unique signature. Example below,

Third constructor Machine(name,code) is called in the first constructor



StringBuilder

StringBuilder s = new StringBuilder();

s.append(“Write a Text”)

.append(“Second text”)

.append(“Third Text”);

This will give appended text of above three

%d is used to replace with int.

Eg System.***out***.printf("Cost is %d;Quantity is %d",5,10);

Output : Cost is 5;Quantity is 10

%s is used to replace with string similar to %d

%f for float. %.2f to round of to two decimal

\n is used for new line and \t is used for tab space

Eg System.***out***.printf("Cost is %d \nQuantity is %d",5,10);

Output

Cost is 5

Quantity is 10

**Inheritance**

**Extends** keywork is used to inherit.

Eg :

public class classA extends classB {

Now an object of Class A can be used to call the methods or variables in classB

}

@Override annotation is used to override the inherited method.

In Eg above if the classB has a methodA and ClassA too has MethodA.

Then for MethodA is ClassA shud have @override to execute MethodA under ClassA.

Use of Final Static Variable

Public final static int constant = 3.14;

static - Only use when a variable which is used globally

final - Only use when you need to declare a value as constant

static final - Only use when a value is globally used and it is a constant.

Interface :

An **interface** is a reference type in **Java**. It is similar to class. It is a collection of abstract methods. A class implements an **interface**, thereby inheriting the abstract methods of the **interface**. Along with abstract methods, an **interface** may also contain constants, default methods, static methods, and nested types

Generics :

Before Java 5, to get the value from the object, the object has to be typecasted as below.

After Java 5, the object can be created generically with specific datatype similar to 2nd example below. This is called as Generics in Java

Before

ArrayList list = new ArrayList();

To retrieva data, the object of list has to be typecasted.

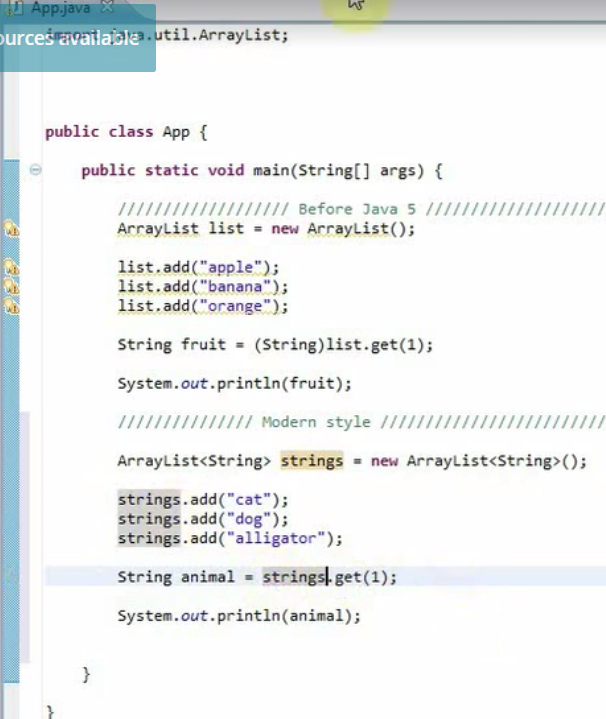
After

ArrayList<String> list = new ArrayList<String>();

No need to type cast

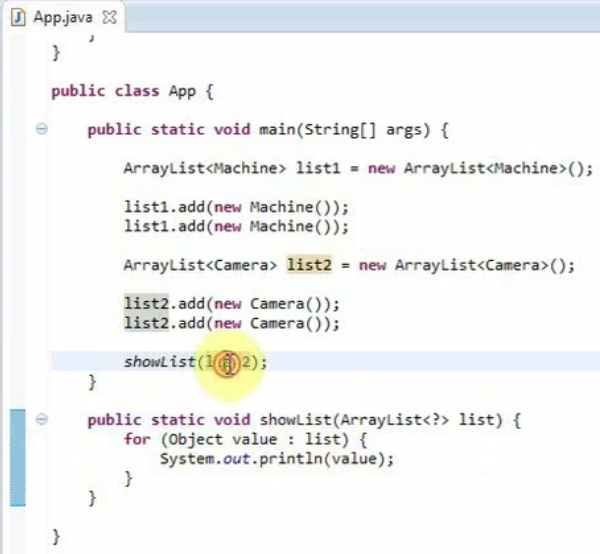
Also, Generics can take multiple arguements

HashMap<Integer, String> mymap1 = new HashMap<Integer, String>();

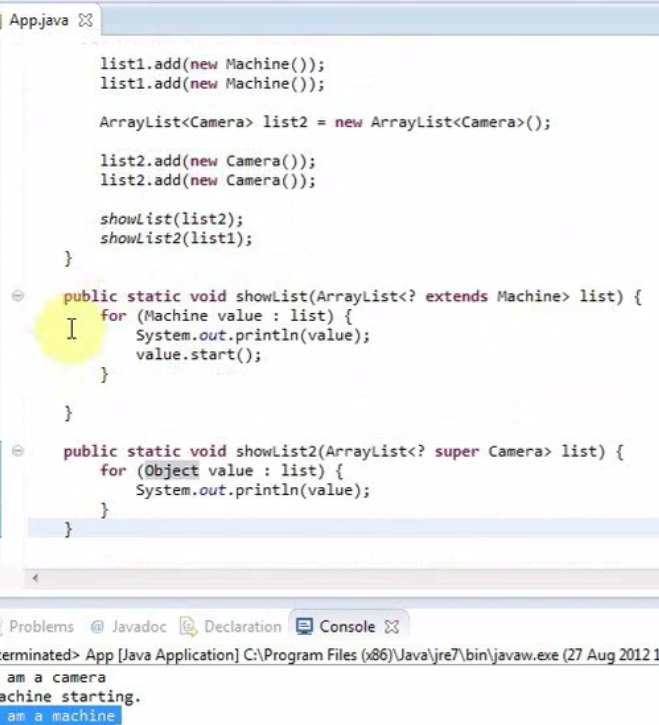


Generics and Wild Card

In the below example “?” is the wild card. This will let the method decide the output based on the input argument. Showlist(list2) and Showlist(list1) both will work. If list1 machine object will be passed as param. If list2 camera object will be passed as param



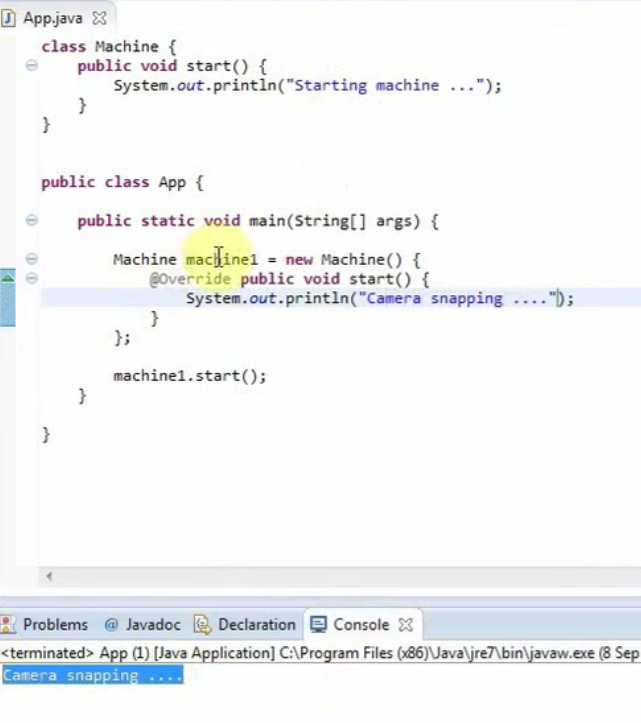
Another case is wild card where ? can extend super class or ? can be super of subclass



Anonymous Classes

This is used to override the class during run time.

An **anonymous class** is defined and instantiated in a single succinct expression using the new operator. While a local **class** definition is a statement in a block of**Java** code, an **anonymous class** definition is an expression, which means that it can be included as part of a larger expression, such as a method call



Checked and UnChecked Exception

Checked Exceptions are Compile time exception

FileNotFound exception, IO Exception are examples of Compile time exception where we read a file and if the file is not present

Unchecked exceptions are run time exception which are not checked during compiling

NullPointer, Arithmatic and ArrayIndexout of bound exception are few of the run time exception

Abstract classes

**Abstract classes** are **classes** that contain one or more **abstract** methods. An**abstract** method is a method that is declared, but contains no implementation.**Abstract classes** may not be instantiated, and require subclasses to provide implementations for the **abstract** methods.

Collections :

ArrayList and Linked List

ArrayList is fast in add when we go sequentially

ArrayList.add(i)

If we have a need to add an item to the list at specified position

LinkedList.add(position, item) is faster.

ArrayList is faster in other operations compared to LinkedList

4 common implementations of Map are

HashMap,TreeMap,LinkedHashMap & HashTable(This permits null)

Set interface extends Collection interface. In a set, no duplicates are allowed. Every element in a set must be unique. You can simply add elements to a set, and duplicates will be removed automatically.

HashSet is Implemented using a hash table. Elements are not ordered. The add, remove, and contains methods have constant time complexity O(1).

TreeSet is implemented using a tree structure(red-black tree in algorithm book). The elements in a set are sorted, but the add, remove, and contains methods has time complexity of O(log (n)). It offers several methods to deal with the ordered set like first(), last(), headSet(), tailSet(), etc.

LinkedHashSet is between HashSet and TreeSet. It is implemented as a hash table with a linked list running through it, so it provides the order of insertion. The time complexity of basic methods is O(1).

HashSet is fastest

**Queue** Interface In **Java**. The **java**.util.**Queue** is a subtype of **java**.util.Collection interface. It is an ordered list of objects with its use limited to inserting elements at the end of list and deleting elements from the start of list i.e. it follows FIFO principle.

**The peek()** This method retrieves the value of the first element of the queue without removing it from the queue. For each invocation of the method we always get the same value and its execution does not affect the size of the queue. **If the queue is empty the peek() method returns null**.

**The element()** This method behaves like peek(), so it again retrieves the value of the first element without removing it. however, if the list is empty element() **throws a NoSuchElementException**.

**The poll()** This method retrieves the value of the first element of the queue by removing it from the queue. . At each invocation it removes the first element of the list and if the list is already empty it returns null but **does not throw any exception**.

**The remove()** This method behaves as the poll() method, so it removes the first element of the list and **if the list is empty it throws a NoSuchElementException**

Eclipse Shortcut

Get a method : Eg : Main method. Type main and ctrl+space

To format method ctrl+shift+F