**The Java Programming Language is a high-level language. It’s characteristics:**

Simple

Object oriented

Distributed

Multithreaded

Dynamic linking

Architecture neutral

Portable

High performance

Robust

Secure

**Wrapper classes are immutable** (non-changeable) because they do not have setters

**NUMBERS CLASSES**

We use a Number object rather than a primitive when:

* + As an argument of a method that expects an object.
  + To use constants defined by the class, such as MIN\_VALUE and MAX\_VALUE.
  + To use class methods for converting values to and from other primitive types.

**A nested class violates the recommendation “low coupling” in class design. Why are nested classes used?**

* It is a way of logically grouping classes that are only used in one place.
* It increases encapsulation.
* Nested classes can lead to more readable and maintainable code.

**INHERITANCE**

A class can be directly derived from only one class ( **Java is a single-inherited OOP language**).

If a class does not have any superclass, then it is implicitly derived from **Object class.**

constructor cannot be inherited ( constructor of super class can not initialize sub-class objects)

Java’s **StringBuffer** and **StringBuilder** classes represent strings that can be dynamically modified.

StringBuffer is threadsafe.

StringBuilder (introduced in 5.0) is not threadsafe.

* Major difference: string builders are not threadsafe.
* If you want multiple threads to have concurrent access to a mutable string, use a string buffer.
* If your mutable string will be accessed only by a single thread, there is an advantage to using a string builder, which will generally execute faster than a string buffer.

The compile-time errors can be:

+ Syntax errors

+ Semantic errors

|  |  |
| --- | --- |
| **Compile-Time Errors** | **Runtime-Errors** |
| These are the syntax errors which are detected by the compiler. | These are the errors which are not detected by the compiler and produce wrong results. |
| They prevent the code from running as it detects some syntax errors. | They prevent the code from complete execution. |
| It includes syntax errors such as missing of semicolon(;), misspelling of keywords and identifiers etc. | It includes errors such as dividing a number by zero, finding square root of a negative number etc. |

**The current object: this**

* The keyword **this** returns the address of the current object.
* This holds the address of the region of memory that contains all of the data stored in the instance variables of current object.
* **Scope of this: this** is created and used just when the member method is called. After the member method terminates **this** will be discarded

the keyword **this** will help distinguish field name and parameter name. **this.x** means that x of this object

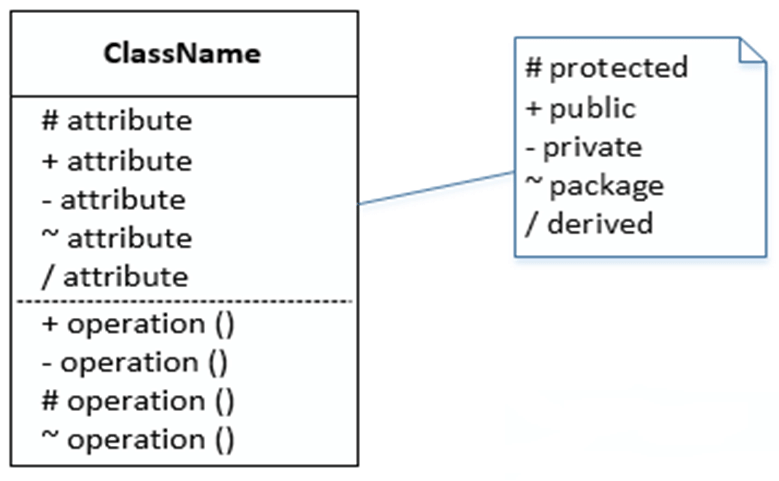
**Passing Arguments a Constructor/Method**

Java uses the mechanism passing by value. Arguments can be:

* + Primitive Data Type Arguments
  + Reference Data Type Arguments (objects)

Text

Description automatically generated



**WHY AND WHEN TO USE INTERFACES?**

* To achieve security - hide certain details and only show the important details of an object (interface).
* Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can implement multiple interfaces

**Why are anonymous classes uesed?**

* Enable you to make your code more concise.
* Enable you to declare and instantiate a class at the same time.
* They are like local classes except that they do not have a name.
* Use them if you need to use a local class only once.

<https://www.geeksforgeeks.org/comparable-vs-comparator-in-java/>

**List Implementing Classes**

ArrayList list= new ArrayList();

for (int i = 101; i <= 110; i++) {

list.add(i);

}

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

System.out.println(list.get(i));

}

//**or using Iterator**

/\*

Iterator iter = list.iterator();

while (iter.hasNext()) {

System.out.println(iter.next());

}

*\*/*

**What are streams?**

A stream is an object managing a data source in which operations such as read data in the stream to a variable, write values of a variable to the stream associated with type conversions are performed automatically. These operations treat data as a chain of units (byte/character/data object) and data are processed in unit-by-unit manner.

**FILE IO**

**Text, UTF, and Unicode**

Java :

* Uses UTF to read/write Unicode
* Helps converting Unicode to external 8-bit encodings and vice versa.

**java.io Package**

* Java treats all data sources ( file, directory, IO devices,…) as **streams**
* The java.io package contains Java APIs for accessing to/from a stream.
* A stream can be a binary stream.
* Binary low-level stream: data unit is a physical byte.
* Binary high-level stream: data unit is primitive data type value or a string.
* Object stream: data unit is an object.
* A stream can be a character stream in which a data unit is an Unicode character.

**Access Text Files**

Character Streams:

* Two ultimate abstract classes of character streams are **Reader** and **Writer**.
* Reader: input character stream will read data from data source (device) to variables (UTF characters).
* Writer: stream will write UTF characters to data source (device).

Binary streams:

Object stream:

**Exception**: Error beyond the control of a program. When an exception occurs, the program will terminate abruptly.

When a program is executing something occurs that is not quite normal from the point of view of the goal at hand. For example:

* + a user might type an invalid filename;
  + An accessed file does not exist of might contain corrupted data;
  + a network link could fail;
  + …

Circumstances of this type are called *exception conditions* in Java and are represented using objects (All exceptions descend from the java.lang.**Throwable**).

**Exceptions are pre-defined data (Exception classes) thrown by JVM and they can be caught by code in the program**

**Kinds of exceptions:**

- **Checked exceptions:**

**+** are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using [try-catch block](http://beginnersbook.com/2013/04/try-catch-in-java/) or it should declare the exception using [throws keyword](http://beginnersbook.com/2013/04/difference-between-throw-and-throws-in-java/), otherwise the program will give a compilation error. It is named as ***checked exception*** because these exceptions are ***checked*** at Compile time.

+ Must be handled by either the try-catch mechanism or the throws-declaration mechanism.

- **Unchecked exceptions**

+ are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn’t handle/declare that exception, the program won’t give a compilation error. Most of the times these exception occurs due to the bad data provided by user during the user-program interaction. It is up to the programmer to judge the conditions in advance, that can cause such exceptions and handle them appropriately. All Unchecked exceptions are direct sub classes of **RuntimeException** class.

+ The right time to deal with runtime exceptions is when you’re designing, developing, and debugging your code. Since runtime exceptions should never be thrown in finished code.