***HTML Applet Tag***

***Applet Tag***

* Used to embed ***JAVA***elements in ***HTML***.
* ***Deprecated*** in ***HTML 4.1*** and ***no longer*** used in ***HTML 5*** .
* It was ***replaced*** by the ***embed*** and ***object tag*** in ***HTML 5***.

***Syntax of Applet Tag***

******

***Attributes used in Applet Tag***

***Code*** : Used to link Java Applet to the HTML File or Document.

***Align*** : used to set the alignment of the Applet element.

***Alt*** : Used for Alternative text.

***Width*** : Used to set the Width.

***Height*** : Used to set the Height.

***Name*** : Used to set the name of the Applet Tag.

***Java Architecture***

* ***refers*** to the ***overall structure*** and components of the Java programming language and its runtime environment.
* encompasses ***both*** the ***development*** and ***execution phases*** of ***Java applications***.

***Java Architecture Diagram***

***A diagram of a computer program

Description automatically generated***

***Components of Java Architecture***

* Java Virtual Machine [***JVM***]
* Java Runtime Environment [***JRE***]
* Java Development Kit [***JDK***]

***Java Virtual Machine*** [***JVM***]

* Java's standout feature is called "***WORA***," which ***stands*** for "***Write Once, Run Anywhere***."
* This ***means*** you can write your Java code just once, and it ***will work*** on ***any*** ***operating system***.
* This magic happens thanks to the "***Java Virtual Machine***" or ***JVM***, a ***crucial part of Java***.
* Think of the ***JVM*** as a sort of ***universal translator*** for ***Java programs***.
* It takes your ***Java code, translates*** it into a form called "***bytecode***," and then ***makes sure*** it ***runs smoothly*** on ***any computer, regardless*** of its ***operating system***.

Here's what the ***JVM does*** in a ***nutshell*** :

***Loading*** : First, it loads your Java code into the computer's memory.

***Verification*** : Then, it checks your code to make sure it's safe and follows the rules.

***Execution*** : After that, it runs your code, creating a runtime environment for it.

***Java Runtime Environment*** *[****JRE****]*

* Is a ***special environment*** where ***Java Programs*** can ***run smoothly***. Imagine it as a ***toolkit*** that takes your Java code, combines it with necessary tools and libraries, and then uses the ***Java Virtual Machine*** [***JVM***] to make your code work.

Here's how it works :

***Integration*** : First, the ***JRE*** takes your Java code and ***combines*** it with all the ***tools*** and ***libraries*** it ***needs***. It's like gathering all the ingredients for a recipe.

***JVM*** : The ***JRE includes a JVM***, which is like a smart interpreter for Java code. It ***reads your code*** and ***turns it*** into something the ***computer can understand***.

Now, let's break down what's ***inside*** the ***JRE*** :

***Deployment*** : This is like the way you serve your Java program. It includes things like Java Web Start, which helps you run programs over the internet.

***User Interface*** : This part makes your program look and feel good. It includes things for graphics, sound, and user interfaces.

***Integration Libraries*** : These are tools that help your Java program talk to other software, like databases or web services.

***Base Libraries*** : Think of these as essential tools. They help with things like handling data, managing files, and connecting to the internet.

***More Libraries*** : There are even more libraries for special tasks, like managing your computer's resources or working with XML.

The ***JRE*** works closely with the ***JVM***, which is like the ***engine*** that ***runs*** your ***Java code***. Here's how they team up :

1. ***Class Loader*** : This part of the JRE loads the pieces of your program that it needs.
2. ***Byte Code Verifier*** : Imagine this as a security check. It makes sure your code is safe and follows the rules before it runs.
3. ***Interpreter*** : This is where the real magic happens. The interpreter reads your code line by line and makes it work, like following a recipe step by step.

***Java Development Kit*** [***JDK***]

* is a ***software development environment*** used in the development of Java applications and applets. Java Development Kit holds JRE, a compiler, an interpreter or loader, and several development tools in it.

Here's what's inside the ***JDK*** :

1. ***JRE*** (***Java Runtime Environment***): This is like the engine that makes Java programs run. It's what you need to run Java software on your computer.
2. ***Compiler*** : Think of this as a translator. It turns your human-readable Java code into a language that computers understand.
3. ***Interpreter/Loader*** : This is like the bridge between your code and the computer
4. ***Development Tools*** : These are like special tools for building and testing Java programs.

***JDK*** comes in ***different versions***, depending on what kind of ***Java applications*** you want to create :

1. ***Standard Edition Java Platform*** : For ***regular*** Java applications.
2. ***Enterprise Edition Java Platform*** : If you're building ***big business applications***.
3. ***Micro Edition Java Platform***: For ***smaller devices*** like ***phones*** and ***smart gadgets***.

Some of the Important Tools Inside the ***JDK*** :

* ***Applet viewer*** : It ***runs and tests Java applets*** (small programs) ***without a web browser***.
* ***Javac*** : This is the ***Java compiler***. It turns your Java code into instructions the computer can follow.
* ***Javadoc*** : It ***automatically creates documentation*** (like a user manual) ***from*** your ***code comments***.
* ***Jar*** : This helps ***package your Java code*** into a neat, ***organized file*** called a ***JAR*** file.
* ***Keytool*** : It's for ***managing security keys*** and certificates to **keep your Java applications secure**.
* ***Jstack*** : If there's ***a problem*** with your ***Java program***, it ***helps you see*** what's ***going on*** by showing the ***stack traces*** of ***different parts*** of your ***program***.
* ***VisualVM*** : A ***visual tool*** that makes it ***easier*** to ***understand*** what your ***Java programs*** are ***doing*** and how they're using computer resources.

***Difference between JVM, JDK, and JRE*** :

***JVM*** (***Java Virtual Machine***) :

* Think of JVM as a ***computer inside your computer***. It ***runs programs written in Java***.
* It ***follows a set of rules*** (a specification) to ***make sure all Java programs run the same way***, no matter what computer they're on.
* It's like a ***universal translator*** that ***understands Java code*** and turns it into actions your computer can take.

***JDK*** (***Java Development Kit***) :

* ***toolbox*** for ***Java programmers***. It ***includes everything*** they need to write, test, and build Java programs.
* Has tool like a ***Compiler***, an ***Interpreter*** and a ***Debugger***.
* it's for ***people*** who not only **run Java programs** but also ***create them***.

***JRE***(***Java Runtime Environment***) :

* a ***package*** that's ***ready to run Java programs***. It ***includes the JVM*** (the virtual computer) and a bunch of ***pre-made tools***.
* It's like having a video game that's ***ready to play***. You don't need to know h***ow the game was built***; you just ***enjoy playing it***.
* is for ***people*** who want to ***use Java programs*** but ***don't need to write*** or ***build*** them.

***JVM*** is the ***engine*** that ***Runs Java***, ***JDK*** is the ***toolkit*** for ***Java Devs*** and ***JRE*** is the ***ready to use package*** for ***Java Users***.

***Inheritance***

* ***Inheritance*** in ***programming*** is like ***building new things*** ***based*** on ***existing blueprints***.
* ***Parent Class*** : This is like the ***original blueprint***. It ***defines how something works***. We also call it the "***base***" or "***superclass***."
* ***Child Class*** : Imagine it as a ***new blueprint*** that ***uses*** the ***original one*** as a ***starting point***. We call it the "***child***" or "***subclass***."
* ***Extends*** : We use the word "***extends***" to say, "***Hey, this new blueprint is based on the old one***."

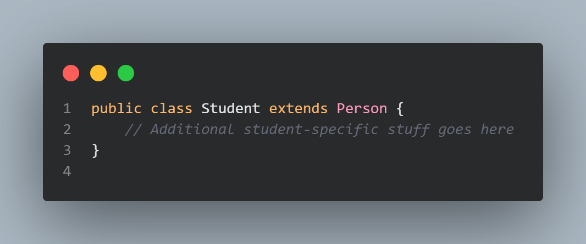
***Benefits of Inheritance***

When we make a ***new blueprint*** (***child class***) based on the ***old one*** (***parent class***)

* We automatically get all the stuff from the old blueprint. So, we don't have to re-create it.
* We only need to change or add things that are different in the new blueprint.

***Example*** :

Imagine we have a blueprint for a "***Person***." Now, we want to make a new blueprint for a "***Student***." We say :



This means our "***Student***" blueprint starts with everything a "***Person***" ***has***.

***Overriding Method Definitions***Sometimes, we want to change how something works in the ***new blueprint*** (***child class***). If we make a ***method*** in the child class that has the same name, same kind of inputs, and does the same job as the one in the ***old blueprint*** (***parent class***), we say we're "***overriding***" it.

For example, if our "***Student***" blueprint wants to do something a bit differently than a "***Person***," we can ***override a method***. This way, when a student does that thing, ***they do it their way***, ***not*** the ***general way*** for everyone (in the "***Person***" blueprint).

***Types of Inheritance***

* ***Single Inheritance*** : It's like having a ***parent*** and a ***child***. A ***child class*** can ***only*** have ***one parent class***.
* ***Multilevel Inheritance*** : Think of it as a ***family tree*** with ***grandparents***, ***parents***, and ***children***. Class ***C extends*** class ***B***, which itself ***extends*** class ***A***.
* ***Hierarchical Inheritance*** : Imagine many ***siblings*** sharing the ***same parent***. ***Classes*** ***B***, ***C***, and ***D*** all ***extend*** the s***ame class A***.
* ***Multiple Inheritance*** : This is like having ***two sets of parents***. But, in ***Java***, it's ***not allowed***. So, ***a child class can't have more than one parent class***.
* ***Hybrid Inheritance*** : It's like a ***mix of different family trees***.
* It's a ***mix of single and hierarchical inheritance***.

***Constructors and Inheritance***

Inheritance is like ***passing down traits from parents to children***. When we create an ***object*** of ***a child class***, something important happens :

* It ***automatically builds*** the ***child object***.
* Then, it goes to the ***parent*** (***superclass***) and ***builds*** that object too.

Think of it like building a tower of blocks, starting from the bottom :

1. First, you build the top block (the child object).
2. Then, you place it on the block below (the parent object).

* But, there's a rule: ***You must put the parent block*** (***superclass constructor***) down ***first***. You ***can't*** build the ***child block*** without the ***parent block underneath***.
* You can talk to the ***parent*** (***superclass***) using "***super***," but you ***can't skip*** ***generations*** and talk to the ***grandparent*** or ***beyond***.

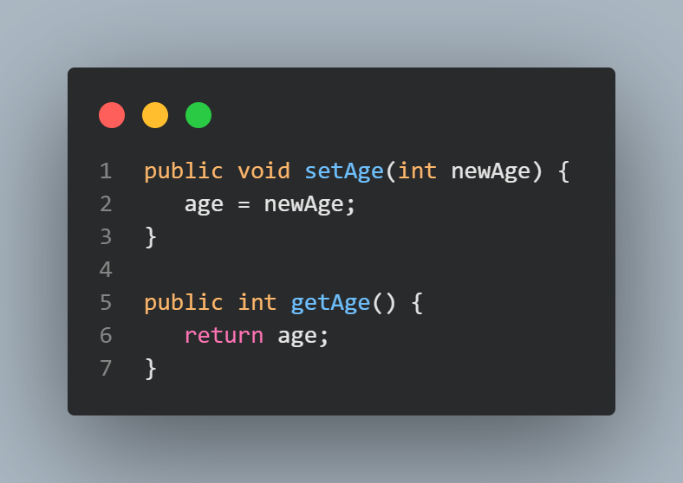
***Encapsulation***

* a ***protective shield*** for your ***data*** and ***actions*** in a ***program***.
* helps protect your data and actions, making sure they're ***accessed*** and ***modified properly***. It's like having a ***locked box*** with ***keys*** for ***controlled access***.

Here's how it works in ***Java*** :

1. You put your ***data*** (like ***name*** and ***age***) in a ***special box*** (a ***class***), and you ***lock the box*** (declare them as "***private***").
2. If someone wants to know your ***age*** or ***name***, you give them a ***key*** (***public methods*** like ***getAge***() and ***getName***()) to safely look at the data inside the locked box.
3. If someone wants to change it. you give them another key (public methods like ***setAge***() and ***setName***())to make sure they do it properly.

Example :



***setAge()*** is like ***giving*** someone a ***key*** to ***update your age***. They can't just ***put anything***; they have to ***follow the rules***.

***getAge()*** is like ***giving*** someone a ***key*** to ***see your age***, but they ***can't change it***.

***Applets***

* In Java, an ***applet*** is a ***special type*** of ***program*** that ***lives inside*** a ***web page*** and ***makes*** things ***happen on the page***.

***Lifecycle***

***A diagram of a life cycle

Description automatically generated***

1. ***Init*** : Gets the applet ready initially. Runs once.
2. ***Start*** : Activates the applet, runs code. Starts after init() and on page reload.
3. ***Stop*** : Pauses the applet. Runs when applet is stopped, minimized, or moved.
4. ***Paint*** : Draws shapes in the applet. Runs after start() and on window resize.
5. ***Destroy*** : When the applet's job is done, it's time to say goodbye.

***Flow of Applet Lifecycle***

These methods are invoked by the browser automatically.

A diagram of a method

Description automatically generated

***Running an Applet***

There are two ways to run an applet :

1. ***HTML File*** :

* Create the applet and compile it.
* Make an HTML file and put the applet code in it. Then Run.

1. ***appletViewer Tool*** :

* Create the applet and compile it.
* Run it with the command: appletviewerAppko.java.

***Event Handling***

Just like we can handle events in AWT or Swing, we can do the same in an applet. Here's a simple example: When you click a button, it prints a message.

Example :



***Displaying Graphics***

In Java, to ***draw things on the screen,*** we use the ***java.awt.Graphics*** class. Here are some common methods :

* ***drawString()***: Draws text.
* ***drawRect()*** : Draws rectangles.
* ***fillRect()***: Fills rectangles.
* ***drawOval()***: Draws ovals.
* ***fillOval()***: Fills ovals.
* ***drawLine()***: Draws lines.
* ***drawImage()***: Draws images.
* ***drawArc()***:Draws arcs.
* ***fillArc()***: Fills arcs.
* ***setColor()***: Sets the color.
* ***setFont()***: Sets the font for text.

***Displaying Images***

In Java applets, you often need to show images for games or animations. To ***display*** an image, you use the ***drawImage***() ***method*** from the ***java.awt.Graphics*** class. Here's how it works :

***drawImage()***: Draws a specified image on the screen at a given position.

To get an image object, you can use the ***getImage()*** method from the ***java.applet.Applet*** class. It helps you load the image you want to display.

Here are some other important methods :

***getDocumentBase()***: ***Gets the URL*** of the ***web page*** where the ***applet*** is ***embedded***.

***getCodeBase()***: Gets the ***base URL*** for the ***applet***.

***Animation***

In Java applets, like in games and animations, you often need to move images around. You use the ***drawImage()*** method from the ***java.awt.Graphics*** class to ***display images***.

One ***important thing*** to note is that the ***drawImage()*** method ***takes*** an ***ImageObserver*** as its ***fourth argument***. The ***good news*** is, you ***can use*** the ***current class object*** as an ***ImageObserver***. Why? ***Because*** the ***Applet class indirectly extends the Component class***, and ***Component implements*** the ImageObserver interface.