\***جاوا کامپایلر چکار می کند؟**

کامپایلر جاوا مسئول ترجمه کدهای جاوا به بایت کد است که یک نمایش سطح پایین(نزدیک به زبان ماشین) و Platform Independent است. در اینجا وظایف اصلی انجام شده توسط کامپایلر جاوا آمده است:

1. **بررسی سینتکس برنامه:** کامپایلر تأیید می کند که کد جاوا از نظر ساختاری و سینتکسی مطابق با قوانین زبان برنامه نویسی جاوا تعریف شده باشد. خطاهایی مانند نزاشتن نقطه ویرگول، پرانتز و یا کلمات کلیدی نادرست را بررسی می کند.

2. **Type Checking:** اطمینان از اینکه متغیرها با یک تایپ صحیح تعریف شده باشند و مقدار اساین شده به آن ها نیز مطابق با تایپ متغیر مربوطه باشد. برای مثال اگر تایپ متغیری از نوع int باشد، تخصیص یک مقدار String به آن سبب ایجاد خطا (Type Error) می شود. و یا عبارات با تایپ مشخص شده در کدهای نوشته شده همخوانی داشته باشد. برای مثال خروجی یک عبارت حسابداری باید با تایپ های عددی سازگار باشد. و یا در متدهای تعریف شده، تایپ پارامترهای ورودی متد با عملیات انجام شده در متد همخوانی داشته و یا خروجی متد با تایپ خروجی که در هنگام ساخت متد مشخص شده است سازگار باشد. و یا اختصاص یک مقدار از جنس یک زیر کلاس به یک متغیر از جنس یک سوپرکلاس به دلیل مفهوم Polymorphism مجاز است. همچنین در بحث مقایسه متغیرها باهم و یا Type Casting ها، استفاده از تایپ های صحیح بررسی و تایید شود. Type Checking در جاوا به شناسایی خطاهای مربوط به تایپ ها در مرحله کامپایل کمک می کند و از ایجاد استثنا در زمان اجرا یا رفتارهای غیرمنتظره جلوگیری می کند و یک محیط برنامه نویسی قوی تر و قابل اعتمادتر را ایجاد می کند.

3. **کامپایل**: کامپایلر کد جاوا را به بایت کد کامپایل می کند که مجموعه ای از دستورالعمل ها است که می تواند توسط ماشین مجازی جاوا (JVM) اجرا شود. این فرآیند کامپایل شامل ترجمه کد سطح بالای جاوا به یک نمایش سطح پایین تر است که می تواند توسط JVM درک و اجرا شود.

4. **گزارش خطا:** اگر کامپایلر در طول فرآیند کامپایل با هر گونه خطا یا هشدار مواجه شود، توسعه دهنده را در مورد مشکلات احتمالی در کد آگاه می کند. این پیام‌ها به توسعه‌دهنده کمک می‌کنند تا قبل از اجرای برنامه، هر گونه مشکلی را شناسایی و برطرف کند.

5. **بهینه سازی**: کامپایلر تکنیک های مختلف بهینه سازی را برای بهبود عملکرد بایت کد تولید شده اعمال می کند. این بهینه‌سازی‌ها شامل constant folding، dead code elimination، inlining و loop transformations می‌شود. هدف تولید بایت کد کارآمدتر است که بتواند توسط JVM سریعتر اجرا شود.

1-4. **Inlining**: زمانی که یک متد در جاوا فراخوانی می شود (Method Invocation or Method Calling)، سبب ایجاد یک بار در سیستم خواهد شد که شامل ارسال مقادیر ورودی به حافظه Stack، ایجاد یک Stack Frame جدید (ساختار تعریف شده در JVM برای مدیریت بهتر اجرای متدها) و تطابق خروجی متد با تایپ مشخص شده در هنگام تعریف متد می باشد. هدف از Inlining کاهش بار ایجاد شده در سیستم با فراخوانی متدها می باشد. برای مثال در inlining به جای فراخوانی متد Add، بدنه اصلی متد یعنی return a+b را جایگزین آن می کند:

public class Example}

public static int add (int a, int b) }

return a + b;

{

public static void main(String[] args) {

int result = add(5, 3); // Method call to add()

System.out.println(result);

}

}

نمونه Inline شده کد بالا:

public class Example }

public static void main(String[] args) {

int result = 5 + 3; // Inlined code

System.out.println(result);

{

{

Inlining سبب بهبود پرفورمنس و کاهش زمان اجرای کد می شود. کامپایلر جاوا در هنگام استفاده از امکان Inlining به نکاتی مانند میزان سادگی متدها و یا فراخوانی کال کردن آن ها و همچنین کانفیگ های مشخص شده در تنظیمات کامپایلر توجه می کند.

2-4. **Constant Folding**: در طول فرآیند کامپایل، کامپایلر جاوا کد را تجزیه و تحلیل می کند و عبارات ثابت را شناسایی می کند. سپس این عبارات را ارزیابی کرده و نتایج محاسبه شده را جایگزین آنها می کند. constant folding با حذف نیاز به انجام محاسبات در زمان اجرا، می تواند عملکرد کد کامپایل شده را بهبود بخشد. برای مثال در کد زیر مقدار result یک مقدار ثابت و مشخص 50 می باشد. بنابراین کد کامپایل شده به طور موثر به "int result = 50;" تبدیل می شود و نیاز به عملیات ضرب در زمان اجرا را از بین می برد.

final int x = 10;

int result = x \* 5; // The expression x \* 5 is a constant expression

System.out.println(result)

constant folding به کاهش مقدار محاسباتی که باید در زمان اجرا انجام شود کمک می‌کند و در نتیجه اجرای کد بالقوه سریع‌تر و کارآمدتر می‌شود. این یکی از چندین تکنیک بهینه سازی است که توسط کامپایلر جاوا برای بهبود عملکرد برنامه های کامپایل شده جاوا استفاده می شود.

3-4. **Dead code elimination:** هنگامی که کامپایلر کد جاوا را تجزیه و تحلیل می کند، بر اساس فلوهای داده ای، تعیین می کند که کدام بخش از کد مرده یا غیرقابل دسترسی است. کد مرده می تواند شامل متغیرهای استفاده نشده، عبارات غیرقابل دسترس یا بلوک های کد باشد که هرگز نمی توانند اجرا شوند. در اینجا چند سناریو وجود دارد که ممکن است کد مرده رخ دهد:

متغیرهای استفاده نشده: اگر متغیری تعریف شده باشد اما هرگز استفاده نشده باشد یا اگر مقدار آن پس از تخصیص استفاده نشود، کامپایلر می تواند آن را شناسایی کرده و آن متغیر و هر گونه تخصیص مرتبط به آن را حذف کند.

کد غیرقابل دسترسی: اگر کامپایلر تشخیص دهد که در طول اجرای برنامه هرگز نمی توان به بخش خاصی از کد دسترسی پیدا کرد، آن کد را مرده تلقی می کند و آن را حذف می کند. برای مثال، پس از یک دستور return، یک دستور نوشته شود.

دستورات شرطی: اگر شرط یک دستور if یا یک حلقه همیشه درست یا نادرست باشد، شاخه یا حلقه مربوطه را می توان به عنوان کد مرده حذف کرد.

4-4. **Loop transformations:** شامل اصلاح ساختار و اجرای حلقه‌ها برای افزایش کارایی و بهره‌برداری از موازی‌سازی است. به عنوان مثال، اگر دو حلقه مجزا داشته باشیم، یکی انجام عملیات روی یک آرایه برای محاسبه مجموع و دیگری برای محاسبه میانگین باشد، آنها را در یک حلقه ترکیب می کند که مجموع و میانگین را با یک بار پاس کردن محاسبه می کند. ویا اگر حلقه‌ای دارید که روی یک آرایه بزرگ تکرار می‌شود، آن را به تکه‌های کوچک‌تر تقسیم کرده و در هر زمان بخشی از آرایه را پردازش می‌کند. از قابلیت های اجرای موازی برای بهبود عملکرد در سیستم های چند هسته ای نیز استفاده می کند.

هنگامی که کد جاوا با موفقیت در بایت کد کامپایل شد، می توان آن را بر روی هر پلتفرمی که یک JVM سازگار نصب کرده است اجرا کرد. JVM مسئول تفسیر یا کامپایل بیشتر بایت کد به کد ماشین است که می تواند توسط OS اجرا شود.

\***کامپایلر جاوا، قبل از کامپایل برنامه چکار می کند؟**

قبل از کامپایل یک برنامه جاوا، کامپایلر جاوا (javac) چندین کار مهم را برای آماده سازی کد برای کامپایل انجام می دهد. در ادامه به مراحل اصلی انجام شده توسط کامپایلر جاوا قبل از فرآیند کامپایل اشاره می شود:

1. **تحلیل واژگانی:** کامپایلر کد را به صورت جریانی از کاراکترها خوانده و تحلیل واژگانی را انجام می دهد. و کد را به نشانه های معناداری مانند کلمات کلیدی، شناسه ها، عملگرها، کلمات و علائم نقطه گذاری تقسیم می کند. این مرحله به ایجاد یک ساختار بیسیک از برنامه کمک می کند.

2. **تجزیه و تحلیل نحوی (Parsing):** کامپایلر نشانه های تولید شده از تحلیل واژگانی را پارس می کند و بررسی می کند که آیا یک ساختار ولید از برنامه مطابق با دستور زبان جاوا وجود دارد. تأیید می کند که کد با قوانین نحوی تعریف شده توسط زبان مطابقت دارد. اگر خطاهای نحوی وجود داشته باشد، کامپایلر خطاهای کامپایل را ایجاد می کند که نشان دهنده مشکلات خاص یافت شده است.

3. **تحلیل معنایی:** پس از تایید سینتکس برنامه، کامپایلر تجزیه و تحلیل معنایی را انجام می دهد تا معنی و صحت کد را بررسی کند. روابط بین شناسه ها، تایپ ها و عبارات را بررسی می کند تا از سازگاری و پایبندی به قوانین زبان اطمینان حاصل کند. این مرحله شامل وظایفی مانند تفکیک نام، بررسی تایپ ها و قوانین و رول های مربوط به دسترسی ها است.

- **نامگذاری صحیح**: کامپایلر ارجاع صحیح به شناسه‌ها، مانند متغیرها، متدها و کلاس‌ها را بررسی می‌کند. این تضمین می کند که شناسه ها قبل از استفاده تعریف شده اند و هرگونه تضاد نامگذاری احتمالی را حل می کند.

- Type Checking: کامپایلر تأیید می کند که عبارات، متغیرها و فراخوانی های متد دارای تایپ های سازگار و مطابق با قوانین تایپ ها در جاوا هستند. این تضمین می کند که عملیات بر روی تایپ های مناسب انجام می شود و هیچ گونه خطای مربوط به تایپ وجود ندارد.

- Scope Rules: کامپایلر امکان مشاهده و دسترسی به متغیرها و سایر انتیتی ها را در قسمت های مختلف کد تعیین می کند. بررسی می کند که متغیرها در اسکوپ مناسب تعریف شده و در محدوده تعریف شده خود استفاده می شوند.

4. **ایجاد یک درختواره سینتکسی:** زمانی که کد مراحل تحلیل واژگانی، نحوی و معنایی را پشت سر گذاشت، کامپایلر یک نمایش میانی از برنامه تولید می کند. این نمایش، اغلب به شکل یک درخت انتزاعی از سینتکس (AST) یا یک نمایش بایت کد میانی، اطلاعات ساختاری و معنایی کد را برای پردازش و بهینه سازی های آینده ذخیره می کند.

**این مراحل اولیه که توسط کامپایلر جاوا انجام می‌شود، اطمینان حاصل می‌کند که کد قبل از شروع به مرحله کامپایل، به خوبی شکل گرفته، از نظر نحوی صحیح و از نظر معنایی معتبر است. اگر هر گونه خطا یا مشکلی در طی این مراحل شناسایی شود، کامپایلر آنها را به عنوان خطا یا اخطار کامپایل گزارش می کند و برنامه نویس را در رفع مشکلات قبل از تولید بایت کد یا خروجی اجرایی راهنمایی می کند.**

**\*چرا نام کلاس ها و فایل ها در جاوا باید یکسان باشد؟**

در جاوا، نامگذاری کلاس جاوا با نام فایلی که در آن تعریف شده یک قرارداد است. این قرارداد یک الزام سختگیرانه نیست که توسط خود زبان جاوا اعمال شود، اما به دلایل مختلف توسط توسعه دهندگان جاوا به طور گسترده دنبال می شود:

1. **سازماندهی کد**: زمانی که نام فایل با نام کلاس مطابقت داشته باشد، **به سازماندهی و مکان یابی کد کمک می کند.** پیدا کردن فایل منبع مربوط به یک کلاس خاص آسان تر می شود، به خصوص در برنامه های بزرگتر با تعداد فایل های زیاد.

2. **خوانایی و قابل فهم**: داشتن نام فایل و نام کلاس مطابقت، **خوانایی و درک کد را بهبود می بخشد**. هنگامی که توسعه دهندگان کد را می خوانند یا با آن کار می کنند، می توانند به سرعت تشخیص دهند که کدام فایل حاوی اجرای یک کلاس خاص است.

3. **کلاس لودرها**: کلاس لودرهای جاوا، که مسئول بارگذاری کلاس ها در زمان اجرا هستند، معمولاً از یک قرارداد نامگذاری خاص برای مکان یابی و بارگذاری کلاس ها از سیستم فایل استفاده می کنند. با پیروی از قرارداد نامگذاری کلاس-فایل، **پیدا کردن و بارگذاری کلاس مناسب را برای کلاس‌لودرها آسان‌تر می‌کند.**

4. **کامپایل و استقرار:** کامپایلر جاوا انتظار دارد که نام کلاس و نام فایل مطابقت داشته باشند. هنگامی که فایل های جاوا را کامپایل می کنید، **شناسایی و کامپایل کلاس صحیح آسان تر می شود.** این تضمین می کند که کامپایلر به راحتی می تواند کلاس صحیح را پیدا و کامپایل کند. به طور مشابه، وقتی برنامه‌های جاوا را بسته‌بندی و **استقرار** می‌دهید، نام‌گذاری مرسوم **فرآیند مکان‌یابی و گنجاندن فایل‌های لازم را ساده می‌کند**.

5. **ابزار و پشتیبانی IDE:** بسیاری از محیط‌های توسعه یکپارچه (IDE) و ابزارهای توسعه برای ارائه ویژگی‌های مفید به قراردادی از مطابقت نام فایل‌ها و کلاس‌ها متکی هستند. به عنوان مثال، **IDE ها می توانند به طور خودکار به فایل منبع زمانی که روی نام کلاس کلیک می کنید، هدایت شوند، یا زمانی که یک کلاس جدید ایجاد می کنید، می توانند نام فایل های مناسب را پیشنهاد کنند.**

توجه به این نکته مهم است که اگرچه پیروی از این کنوانسیون یک عمل خوب است، اما **اجباری نیست.** شما می توانید چندین کلاس را در یک فایل تعریف کنید، به شرطی که فقط یکی از آنها بصورت Public اعلام شود و نام آن با نام فایل مطابقت داشته باشد. با این حال، به طور کلی توصیه می شود برای حفظ وضوح و سازماندهی کد، یک کلاس Public در هر فایل داشته باشید.

IntelliJ IDEA، مانند بسیاری دیگر از IDE های جاوا، از این قرارداد پیروی می کند که نام کلاس جاوا باید با نام فایلی که در آن تعریف شده است مطابقت داشته باشد. اگر از این قرارداد منحرف شوید و کلاسی با نامی متفاوت از فایل دارید، IntelliJ IDEA ممکن است خطا یا هشداری را نشان دهد. دلیل این امر ان است که **IntelliJ IDEA کد شما را تجزیه و تحلیل می کند و ویژگی هایی مانند Navigation Code، refactoring و Completion Code را بر اساس فرضیات و قراردادها ارائه می دهد. زمانی که نام کلاس با نام فایل مطابقت نداشته باشد، ممکن است باعث سردرگمی شود و توانایی IDE برای ارائه پیشنهادات دقیق و انجام عملیات خاص را مختل کند.**

برای جلوگیری از خطاها یا هشدارهای مربوط به عدم تطابق نام کلاس و فایل در IntelliJ IDEA، به طور کلی توصیه می شود از قرارداد نامگذاری فایل مشابه با کلاس پیروی کنید. با این حال، ا**گر عمداً به نام کلاس دیگری نیاز دارید، می‌توانید اخطارهای مربوطه را در تنظیمات IDE یا Annotationها، Disable کنید اما توصیه می‌شود که از چنین استثناهایی به‌دقت و فقط در صورت لزوم استفاده کنید.**

**\*زبان برنامه نویسی** **Groovy** چیست؟

گرووی یک زبان برنامه نویسی نسبتاً پر‌کاربرد شیء‌گرا و داینامیک است که برای ماشین مجازی جاوا JVM مورد استفاده قرار می‌گیرد و در هرجایی از جاوا قابل اجرا می‌باشد. زبان Groovy برای توسعه‌دهندگان پلتفرم جاوا معمولا به عنوان یک زبان اسکریپت‌نویسی استفاده می شود. Groovy به طور یکپارچه با بایت‌کد جاوا کار کرده و دارای سینتکس مشابه با جاوا می‌باشد.

**مهمترین کاربرد‌های Groovy**

* ترکیب ماژول‌های [جاوا](https://maktabkhooneh.org/mag/what-is-javascript-and-what-is-it-for/)
* گسترش برنامه‌های موجود جاوا
* نوشتن برنامه‌های جدید در جاوا
* مناسب برای اسکریپت‌نویسی با سینتکس آسان و قدرتمند
* توسعه داده شده برای ماشین مجازی جاوا
* مناسب برای استفاده از داده‌های استاتیک و داینامیک
* تلفیق شده با ویژگی [زبان‌ پایتون](https://maktabkhooneh.org/mag/python-programming-language/) و Ruby
* دارای قابلیت‌ تبدیل شدن به (DSL (Domain Specific Language و برنامه‌نویسی متا
* مناسب برای نوشتن اسکریپت‌های ساخت (build) در ابزار Gradle
* امکان برقراری ارتباط با برنامه‌های جاوا و کتابخانه‌های آن
* امکان یادگیری آسان و بدون دردسر
* بهره‌بری از قابلیت Functional Programming

این زبان ویژگی های زیادی دارد. مانند اینکه عملگرها بصورت تابع فراخوانی می شوند. در برنامه نویسی Groovy برای تعریف یک متغیر نیازی نیست که حتماً نوع آن را مشخص کنیم. با استفاده از def می توانیم متغیر تعریف کنیم. **کلاس‌های جاوا در گرووی قابل استفاده هستند و از آنجایی که کلاس‌های گرووی به بایت‌کد جاوا ترجمه می‌شوند می‌توان از کلاس‌های Groovy در جاوا نیز استفاده کرد.** برای استفاده از gradle هم استفاده می شود (گریدل در اندروید استودیو به برنامه نویسان کمک می کند تا زمان ایجاد مجدد apk کارشان راحت تر، سریع تر و با خطای کمتری باشد و برای اعمال تغییرات جدید توسط برنامه نویس نیاز به تکرار و انجام کارهای مکرر نباشد).

**\*GraalVM چیست؟**

ممکن است بخواهیم اپلیکیشنی را بنویسیم که در آن از چندین زبان برنامه‌نویسی استفاده شده باشد و نیاز داشته باشیم تا با استفاده از یک ماشین مجازی، تمامی آن‌ها را به صورت هم‌زمان اجرا کنیم. [GraalVM](http://www.graalvm.org/) یک ماشین مجازی همه‌کاره است که برنامه‌های نوشته شده به زبان‌های JavaScript ،Python 3 ،Ruby و R را اجرا می‌کند. این ماشین مجازی (Virtual Machine) همچنین می‌تواند زبان‌های مبتنی بر JVM مثل Java ،Scala و Kotlin و حتی زبان‌های مبتنی بر LLVM همچون C و ++C را نیز اجرا کند.

GraalVM مرز انتزاعی موجود بین زبان‌های برنامه‌نویسی مختلف را از بین می‌برد و قابلیت اجرای برنامه‌ها در یک زمان را فراهم می‌کند. این ماشین‌مجازی هم می‌تواند به صورت جداگانه اجرا شود و هم می‌تواند در متن برنامه‌ها یی مثل Node.js ،OpenJDK ،Oracle Database و یا MySQL قرار داده شود.

GraalVM به شما این امکان را می‌دهد تا اپلیکیشن‌های چندزبانه بنویسید و بهترین زبان را برای کار خودتان انتخاب کنید. در واقع، فایل‌های نِیتیو که از قبل توسط GraalVM کامپایل می‌شوند، **منجر به کاهش زمان لانچ شده** و در نهایت **میزان مصرف حافظه اپلیکیشن‌های مبتنی بر JVM را کاهش می‌دهد**. با استفاده از GraalVM می توانیم برنامه‌های خود را سریع‌تر اجرا کرده و فایل‌های نیتیو ایجاد یا کدهای خود را توسعه داده و بهبود بخشیم.

**\*چه ابزارهای Version Controlای وجود دارند؟**

چندین ابزار محبوب برای کنترل نسخه در جاوا وجود دارد. در اینجا برخی از پرکاربردترین آنها آورده شده است:

1. Git: **Git** یک سیستم کنترل نسخه توزیع شده است که به طور گسترده در جامعه توسعه نرم افزار استفاده می شود. تنها مختص جاوا نیست اما به طور گسترده برای کنترل نسخه در پروژه های جاوا استفاده می شود. Git به شما این امکان را می دهد که تغییرات در پایگاه کد خود را ردیابی کنید، با توسعه دهندگان دیگر همکاری کنید و برنچ های مختلف کد خود را مدیریت کنید. برخی از پلتفرم های میزبانی Git محبوب عبارتند از GitHub، GitLab و Bitbucket.

2. Subversion (SVN): **Subversion** یک سیستم کنترل نسخه متمرکز است که به طور گسترده در پروژه های جاوا نیز استفاده می شود. ویژگی هایی مشابه Git ارائه می دهد اما از معماری متفاوتی پیروی می کند. با SVN می توانید تغییرات را ردیابی کنید، برنچ ها را مدیریت کنید و با توسعه دهندگان دیگر همکاری کنید. SVN برای مدت طولانی وجود داشته است و به دلیل پایداری و سهولت استفاده شناخته شده است.

3. Mercurial: **Mercurial** یکی دیگر از سیستم های کنترل نسخه توزیع شده مشابه Git است. این یک رابط کاربری ساده و شهودی ارائه می‌دهد که شروع کار با کنترل نسخه را برای توسعه‌دهندگان آسان‌تر می‌کند. اگرچه مرکوریال به اندازه Git یا SVN به طور گسترده مورد استفاده قرار نمی گیرد، اما هنوز یک گزینه مناسب برای کنترل نسخه در پروژه های جاوا است.

4. Apache Maven: **Apache Maven** یک ابزار اتوماسیون Build و مدیریت Dependencyها است، اما برخی از قابلیت های کنترل نسخه را نیز ارائه می دهد. Maven به شما امکان تعریف و مدیریت وابستگی های پروژه، از جمله نسخه های خاص کتابخانه ها و فریم ورک ها را می دهد. این می تواند به اطمینان از ساخت های سازگار در محیط های مختلف کمک کند و مدیریت وابستگی ها را در پروژه های جاوا ساده کند.

5. Apache Ant: **Apache Ant** یک ابزار اتوماسیون Build شبیه به Maven اما با رویکردی متفاوت است. در حالی که Ant قابلیت کنترل نسخه صریح را ارائه نمی دهد، می توان آن را با سایر سیستم های کنترل نسخه ادغام کرد تا فرآیندهای ساخت را خودکار کند، از جمله واکشی نسخه های خاصی از کد از یک مخزن.

اینها تنها چند نمونه از ابزارهای کنترل نسخه هستند که معمولاً در توسعه جاوا استفاده می شوند. انتخاب ابزار به نیازهای خاص شما و ترجیحات تیم توسعه شما بستگی دارد. Git در سال های اخیر محبوبیت قابل توجهی به دست آورده است و اغلب به دلیل انعطاف پذیری، پشتیبانی گسترده جامعه و ادغام با پلتفرم های میزبانی محبوب مانند GitHub، ابزار انتخابی برای بسیاری از پروژه های جاوا است.

**\*چرا گیت محبوب است؟**

Git به چند دلیل به عنوان یک ابزار کنترل نسخه بسیار محبوب شده است:

1. **Distributed Version Control**: یک سیستم کنترل نسخه توزیع شده (DVCS) است، به این معنی که هر کاربر یک کپی کامل از کل repository ، از جمله کل History را دارد. این به توسعه دهندگان اجازه می دهد تا به صورت آفلاین کار کنند، تغییرات را به صورت لوکالی انجام دهند و به طور یکپارچه با دیگران همکاری کنند. همچنین انعطاف پذیری را در مدیریت برنچ ها و مرج تغییرات فراهم می کند و کار بر روی فیچرهای مختلف را به طور همزمان آسان تر می کند.

2. **سرعت و عملکرد**: Git به دلیل سرعت و عملکرد خود، حتی با کدهای بزرگ و هیستوری های گسترده، شناخته شده است. این کار را از طریق ساختارهای داده و الگوریتم‌های کارآمد خود به دست می‌آورد و به عملیات‌هایی مانند commit، شاخه‌بندی و مرج اجازه می‌دهد در مقایسه با برخی دیگر از سیستم‌های کنترل نسخه، سریع باشند.

3. **Branching و Merging:** Git قابلیت‌های قدرتمند برنچ بندی و مرج تغییرات را فراهم می‌کند و ایجاد و مدیریت برنچ ها را برای فیچر‌های مختلف و فیکس کردن باگ ها را آسان می‌کند. برنچ بندی در Git سبک است و نیازی به کپی کردن کل پایگاه کد ندارد، به توسعه‌دهندگان اجازه می‌دهد تا به سرعت برنچ هایی ایجاد کنند و بدون زحمت بین آن‌ها جابه‌جا شوند. الگوریتم های ادغام Git نیز قوی هستند و می توانند سناریوهای ادغام پیچیده را به راحتی مدیریت کنند.

4. **انجمن و اکوسیستم**: گیت دارای یک جامعه گسترده و فعال از توسعه دهندگان است که منجر به اکوسیستم غنی از ابزارها، برنامه های افزودنی و ادغام شده است. این شامل پلتفرم‌های میزبانی محبوب مانند GitHub، GitLab و Bitbucket است که یک رابط کاربر پسند، issue tracking ، pull requests و سایر ویژگی‌های همکاری را ارائه می‌کنند. پشتیبانی گسترده جامعه و مستندات موجود یادگیری و استفاده مؤثر از Git را برای توسعه‌دهندگان آسان‌تر می‌کند.

5. **انعطاف پذیری و سازگاری**: Git به گونه ای طراحی شده است که انعطاف پذیر و سازگار با گردش کار و محیط های مختلف توسعه باشد. با پروژه های کوچک و بزرگ، از پروژه های شخصی کوچک گرفته تا برنامه های بزرگ سازمانی، به خوبی کار می کند. Git همچنین با سیستم عامل های مختلف (ویندوز، macOS، لینوکس) سازگار است و می تواند به راحتی با سایر ابزارهای توسعه و سیستم های ساخت ادغام شود.

6**. متن باز**: Git خود یک پروژه منبع باز است، به این معنی که کد منبع آن به صورت رایگان در دسترس است و می تواند توسط جامعه تغییر و گسترش یابد. این باز بودن، پذیرش گسترده را تشویق کرده و به بهبود و نوآوری مداوم در Git کمک کرده است.

این عوامل در مجموع به محبوبیت Git به عنوان یک ابزار کنترل نسخه کمک کرده اند. ماهیت توزیع شده، عملکرد، قابلیت های برنچ بندی و مرج، پشتیبانی از جامعه و سازگاری آن را به استاندارد واقعی برای کنترل نسخه در بسیاری از پروژه های توسعه نرم افزار، از جمله پروژه های موجود در اکوسیستم جاوا تبدیل کرده است.

**\*نحوه کار گیت**:

از گیت می توان هم از Interface آن استفاده کرد و هم از دستورات کامندی آن. تغییرات اعمال شده در فایل با دستور کامیت بر روی سیستم کاربر تایید می شوند، سپس برای اینکه تغییرات در دسترس دیگر همکاران پروژه قرار گیرد می توان از دستور Push استفاده کرد. همچنین قبل از اعمال تغییرات بهتر است با دستور Pull تغییرات قبلی اعمال شده توسط همکاران دیگر را دریافت کرد تا از ایجاد تناقضات کدی جلوگیری شود. ریپازیتوری از قبل ایجاد شده و پروژه می توان با دستور Clone وارد سیستم لوکال کاربر شود و ارتباط با پروژه برقرار شود. می توان برنچ های مختلفی جهت دسترسی برنامه نویسان مختلف ایجاد کرد و سپس تغییرات از هر برنچ به شاخه Main مرج شود. و یا در پروژه های بزرگ تر حتی می توان به ازای افراد مختلف ریپازیتوری های مختلف ایجاد کرد و سپس تغییرات هر یک را به ریپازیتوری اصلی اعمال کرد. اگر تناقضاتی ناشی از کار کردن همزمان دو دولوپر بر روی یک کد یکسان ایجاد شود با استفاده از تکنیک های رفع تناقض می توان مشکلات تداخل میان نسخه های مختلف را رفع کرد.

**\*گیت هاب چیت؟**

GitHub یک بستر میزبانی مبتنی بر وب برای کنترل و همکاری نسخه با استفاده از GIT است. این یک رابط کاربر پسند و طیف وسیعی از فیچرها را ارائه می دهد که کار افراد و تیم ها را بر روی پروژه های توسعه نرم افزار آسان تر می کند.

در اینجا برخی از جنبه های کلیدی GitHub آورده شده است:

1. کنترل نسخه: GitHub در بالای Git ساخته شده است ، یک سیستم کنترل نسخه توزیع شده. این امکان را به توسعه دهندگان می دهد تا مخازن را برای ردیابی تغییرات در پایگاه کد خود ایجاد کنند. با Git می‌توانید تغییرات را انجام دهید، شاخه‌هایی را برای ویژگی‌ها یا آزمایش‌های مختلف ایجاد کنید، تغییرات را از شاخه‌های مختلف ادغام کنید و به نسخه‌های قبلی کد خود بازگردید.

2. همکاری: GitHub ویژگی هایی را فراهم می کند که همکاری بین توسعه دهندگان را امکان پذیر می کند. چندین کاربران می توانند به طور همزمان روی همان پروژه کار کنند ، و هماهنگی تلاش ها و بررسی کد یکدیگر را آسان می کنند. GitHub ابزارهایی مانند pull requests ، بررسی کد و issue tracking را ارائه می‌دهد که ارتباط و همکاری را در یک تیم توسعه تسهیل می‌کند.

3. میزبانی مخزن: GitHub میزبان مخازن GIT است و باعث می شود تا با اتصال به اینترنت ، ذخیره و دسترسی به کد خود را آسان کنید. این یک مکان متمرکز برای پایگاه کد شما فراهم می کند و به شما امکان می دهد به راحتی کار خود را با دیگران به اشتراک بگذارید و با توسعه دهندگان در سراسر جهان همکاری کنید.

4- برنامه نویسی جامعه و اجتماعی: GitHub دارای جامعه پر جنب و جوش از توسعه دهندگان است که از همکاران فردی گرفته تا سازمان های بزرگ. این امکان را به توسعه دهندگان می دهد تا پروژه های منبع باز را کشف و مشارکت کنند ، کار خود را به اشتراک بگذارند و تخصص خود را به نمایش بگذارند. GitHub با ارائه ویژگی هایی مانند forking مخازن، ایجاد شاخه ها، و ارسال درخواست های کشش برای ایجاد تغییرات در پروژه اصلی، کدگذاری اجتماعی را تقویت می کند.

5. مدیریت پروژه: GitHub ابزارهای مدیریت پروژه را برای کمک به توسعه دهندگان سازماندهی کار خود ارائه می دهد. می‌توانید داشبورد پروژه ایجاد کنید، مسائل و درخواست‌های ویژگی‌ها را پیگیری کنید، وظایفی را به اعضای تیم اختصاص دهید و نقاط عطف را برای مدیریت پیشرفت پروژه‌های توسعه خود تعیین کنید.

6. یکپارچه‌سازی و توسعه‌پذیری: GitHub با ابزارها و سرویس‌های توسعه مختلف ادغام می‌شود و امکان اتوماسیون و ادغام یکپارچه در جریان‌های کاری موجود را فراهم می‌کند. این برنامه از ادغام با ابزارهای ادغام و استقرار مداوم (CI/CD) ، ابزارهای تجزیه و تحلیل کد ، چارچوب های آزمایش و موارد دیگر پشتیبانی می کند. علاوه بر این ، GitHub یک API را فراهم می کند که به توسعه دهندگان اجازه می دهد تا عملکرد خود را گسترش داده و ادغام های سفارشی ایجاد کنند.

GitHub به دلیل سهولت استفاده، ویژگی‌های همکاری قوی، پشتیبانی گسترده جامعه و قابلیت‌های یکپارچه‌سازی، محبوبیت زیادی در جامعه توسعه نرم‌افزار به دست آورده است. این به یک پلتفرم برای میزبانی و به اشتراک گذاری کد، کمک به پروژه های منبع باز و همکاری در پروژه های توسعه نرم افزار تبدیل شده است.

**\*تفاوت ادیشن های مختلف جاوا در چیست؟**

جاوا یک زبان برنامه نویسی است که دارای نسخه های مختلفی است که هر کدام اهداف خاصی را دنبال می کنند و محیط های مختلفی را هدف قرار می دهند. نسخه های اصلی جاوا:

1. Java SE (نسخه استاندارد): این رایج ترین و پرکاربردترین نسخه جاوا است. کتابخانه های اصلی، پشتیبانی از زبان های مختلف و ابزارهای لازم برای توسعه برنامه های کاربردی همه منظوره را فراهم می کند. Java SE برای برنامه های دسکتاپ، توسعه سمت سرور و اپلیکیشن های standalone مناسب است. و شامل بخش هایی مانند کیت توسعه جاوا (JDK)، محیط اجرای جاوا (JRE) و ماشین مجازی جاوا (JVM) است.

2. Java EE (Enterprise Edition): Java EE یک توسعه Java SE است که به طور خاص برای توسعه برنامه های کاربردی در سطح سازمانی طراحی شده است. مجموعه ای از API ها و محیط های زمان اجرا را برای ساختن برنامه های کاربردی مقیاس پذیر، توزیع شده و تراکنشی فراهم می کند. Java EE شامل ویژگی هایی مانند Java Servlets، JavaServer Pages (JSP)، Enterprise JavaBeans (EJB)، Java Message Service (JMS) و Java Persistence API (JPA) است. جاوا EE توسط جاکارتا EE جایگزین شده است، که نتیجه انتقال Java EE به بنیاد Eclipse است.

3. Java ME (Micro Edition): Java ME یک نسخه حذف شده از جاوا است که در درجه اول برای توسعه برنامه های کاربردی در دستگاه های دارای محدودیت منابع مانند تلفن های همراه، سیستم های جاسازی شده و دستگاه های IoT (اینترنت اشیا) استفاده می شود. Java ME زیرمجموعه ای از Java SE API و مجموعه ای از پروفایل های اضافی بهینه سازی شده برای دستگاه های کوچک را فراهم می کند. این برنامه توسعه دهندگان را قادر می سازد تا برنامه ها و خدمات تلفن همراه را ایجاد کنند که می توانند بر روی طیف گسترده ای از دستگاه ها اجرا شوند.

4. Java FX: JavaFX یک پلت فرم برای ساخت برنامه های کاربردی دسکتاپ غنی و برنامه های کاربردی اینترنتی غنی (RIA) است. مجموعه‌ای از کتابخانه‌ها و ابزارها را برای ایجاد رابط‌های کاربری جذاب، برنامه‌های چند رسانه‌ای و برنامه‌های مبتنی بر داده فراهم می‌کند. JavaFX از زمان جاوا 8 با Java SE همراه شده است و به عنوان فریم ورک پیشنهادی برای ساخت برنامه های دسکتاپ در جاوا در نظر گرفته می شود.

**\*مقایسه امکانات نسخه های مختلف جاوا:**

در اینجا به مقایسه برخی از ویژگی های کلیدی معرفی شده در نسخه های مختلف جاوا می پردازیم:

جاوا 8:

- Lambda Expressions: لامبدا را معرفی کرد که به توسعه دهندگان این امکان را می دهد تا کد مختصر و رساتر بنویسند.

- Streams API: یک API جدید برای کار با کالکشن ها و انجام عملیات بصورت بالک، مانند filtering, mapping و کاهش عناصر ارائه می دهد.

- Date/Time API : به محدودیت های کلاس های قبلی تاریخ و تقویم اضافه شد.

- Default Methods: اینترفیس‌ها مجاز به پیاده‌سازی روش‌های مشخص هستند که امکان تکامل آسان‌تر APIها را فراهم می‌کند.

- Optional Class: یک کلاس جدید برای مدیریت مقادیر null و کاهش NullPointerExceptions معرفی کرد.

جاوا 9:

- Module System (جاوا پلتفرم ماژول سیستم): یک سیستم ماژول برای encapsulate کردن کد و وابستگی ها، بهبود ماژولار بودن و قابلیت نگهداری را معرفی کرد.

- JShell (حلقه Read-Eval-Print): یک ابزار خط فرمان تعاملی برای اجرای قطعه کد جاوا و استفاده از API ها ارائه می کند.

- Process API Updates: API های جدیدی را برای تعامل با فرآیندهای سیستم عامل معرفی کرد که امکان کنترل و مدیریت بهتر فرآیندها را فراهم می کند.

- Private Methods in Interfaces: Interfaces ها مجاز به داشتن متدهای خصوصی هستند که سازماندهی کد و استفاده مجدد را تسهیل می‌کنند.

جاوا 10:

- Local-Variable Type Inference: کلمه کلیدی var را برای استنتاج نوع معرفی کرد که به توسعه دهندگان اجازه می دهد تا متغیرهای محلی را بدون تعیین صریح نوع متغیر اعلام کنند.

- GC کامل موازی برای G1: با فعال کردن پردازش موازی در طول جمع‌آوری کامل زباله، زباله‌گیر اول (G1) را بهبود بخشید.

- Application Class-Data Sharing: مکانیزمی را برای بهبود زمان راه اندازی و کاهش ردپای حافظه با اشتراک گذاری ابرداده های کلاس مشترک در نمونه های JVM معرفی کرد.

جاوا 11:

- HTTP Client API: یک API سرویس گیرنده HTTP استاندارد جدید را برای جایگزینی کلاس HttpURLConnection قدیمی معرفی کرد.

- Local-Variable Syntax for Lambda Parameters: کلمه کلیدی var را به پارامترهای لامبدا گسترش داد و در هنگام استفاده از عبارات لامبدا، کدهای طولانی و زیاد را کاهش داد.

- Flight Recorder: جاوا Flight Recorder (JFR) را به صورت رایگان در OpenJDK در دسترس قرار داد و قابلیت های تشخیصی و نمایه سازی کم سربار را ارائه کرد.

جاوا 12:

- Switch Expressions: عبارت switch را بهبود بخشید تا امکان استفاده از آن به عنوان یک عبارت را فراهم کند و کد مختصرتر و گویاتری ارائه دهد.

- Compact Number Format: کلاس جدیدی به نام NumberFormat را برای قالب بندی اعداد به شکل فشرده و قابل خواندن توسط انسان معرفی کرد.

جاوا 13:

- Text Blocks

- Switch Expressions (بهبود): ویژگی عبارات سوئیچ را با پیشرفت های اضافی، از جمله عبارات عملکرد و ساده کردن عبارات، گسترش داد.

جاوا 14:

- Records: یک ویژگی زبان جدید به نام رکوردها را معرفی کرد که روشی مختصر برای اعلام کلاس هایی که عمدتاً برای ذخیره داده ها استفاده می شوند ارائه می دهد.

- تطبیق الگو برای instanceof: عملگر instanceof را برای ساده‌سازی بررسی نوع و conditional casting تقویت کرد.

- NullPointerException های مفید: پیام های NullPointerException را با ارائه اطلاعات دقیق تر در مورد مرجع تهی که باعث استثنا شده است، بهبود بخشید.

**\*مطالعه راجع به JDK, JRE و JVM**

در جاوا، اصطلاحات JDK، JRE و JVM به اجزای مختلف پلتفرم جاوا اشاره دارد.

1. JDK (کیت توسعه جاوا) : JDK یک کیت توسعه نرم افزار است که ابزارهای لازم برای دولوپ ، کامپایل و دیباگینگ برنامه های جاوا را فراهم می کند. که شامل کامپایلر جاوا (Javac)، محیط RunTime (java)، و ابزارها و library های مختلف توسعه است. JDK در درجه اول توسط توسعه دهندگان نرم افزاری که کد جاوا را می نویسند استفاده می شود. و شامل JRE (در زیر توضیح داده شده) نیز هست. از جمله Toolsهایی که در JDK قرار دارند عبارتند از کامپایلر جاوا (Javac)، محیط RunTime (java) که برای لود بایت کدها و ران کردن برنامه های جاوا روی JVM کاربرد دارد، Java Archive Tool (jar) که فایل های برنامه را فشرده کرده و در قالب یک فایل Jar در می آورد و به این ترتیب فرایند انتقال فایل ها و استقرار را ساده تر می کند، Java Debugger (jdb) که برای ایجاد BreakPointها، رصد متغیرها و بررسی گام به گام کد و تشخیص و شناسایی ایشوهای موجود در کد به دولوپرها کمک می کند، Java Documentation Generator (javadoc) که با استفاده از انوتیشن Javadoc کامنت های وسط برنامه را مدیریت کرده و می تواند آن ها را به یک فایل HTMLای تبدیل کند. Java Development Tools (javap, javah, javafxpackager) و دیگر ابزارها...

JDK همچنین شامل کتابخانه های Java API (رابط برنامه نویسی برنامه) است که کلاس ها و متدهای از پیش ساخته شده را برای یک سری کارهای رایج مانند فایل هندلینگ، شبکه سازی و توسعه رابط کاربری ارائه می دهد.

2. \*\*JRE (Java Runtime Environment): JRE یک محیطRuntime است که به شما امکان اجرای برنامه های جاوا را می دهد. که شامل ماشین مجازی جاوا (JVM) و مجموعه ای از کتابخانه ها و منابع مورد نیاز برای اجرای برنامه های جاوا است. JRE برای کاربران نهایی در نظر گرفته شده است که می خواهند برنامه های جاوا را بدون نیاز به توسعه آنها اجرا کنند. JRE محیط Runtime و کتابخانه های لازم را برای اجرای Bytecode Java ، که فرم کامپایل شده کد منبع جاوا است ، فراهم می کند.

JRE شامل ابزارهای توسعه (مانند کامپایلر جاوا) موجود در JDK نمی شود. این یک بسته کوچکتر است که برای اجرای برنامه های جاوا به جای توسعه آنها طراحی شده است.

3. JVM (Java Virtual Machine): JVM یک ماشین مجازی است که یک محیط برای Excuteکردن برنامه های جاوا فراهم می کند. یک جزء حیاتی از پلتفرم جاوا است که وظیفه تفسیر و اجرای بایت کد جاوا را بر عهده دارد. JVM با جزئیات سخت افزار و سیستم عامل ارتباط برقرار کرده و به برنامه های جاوا اجازه می دهد به طور مداوم در پلتفرم های مختلف اجرا شوند.

هنگامی که یک برنامه جاوا را اجرا می کنید، JVM مسئول بارگذاری و اجرای بایت کد تولید شده توسط کامپایلر جاوا است. مدیریت حافظه، garbage Collector و جزئیات Runtime مانند thread management ، بررسی‌های امنیتی و exception handling را ارائه می‌دهد. JVM تضمین می کند که برنامه های جاوا با بهینه سازی اجرای کد و اعمال محدودیت های Runtime مختلف به طور کارآمد و ایمن اجرا می شوند.

توجه به این نکته حائز اهمیت است که پیاده سازی های مختلفی از JVM وجود دارد که توسط فروشندگان مختلف ارائه شده است ، اما همه آنها به مشخصات دستگاه مجازی جاوا پایبند هستند و از سازگاری با زبان جاوا و کد استفاده می کنند. به طور خلاصه، JDK برای توسعه برنامه های جاوا استفاده می شود و شامل JRE است که برای اجرای برنامه های جاوا استفاده می شود. JDK و JRE هر دو به JVM متکی هستند که بایت کد جاوا را اجرا می کند و محیط زمان اجرا لازم را برای برنامه های جاوا فراهم می کند.

**\*JIT چیست؟**

JIT مخفف Just-In-Time compilation است و این تکنیکی است که توسط دستگاه مجازی جاوا (JVM) برای بهبود عملکرد برنامه های جاوا در زمان اجرا استفاده می شود. هنگامی که یک برنامه جاوا اجرا می شود، ابتدا از کد قابل خواندن توسط انسان به بایت کد مستقل از پلت فرم کامپایل می شود. سپس این بایت کد توسط JVM تفسیر و اجرا می شود. در رویکرد تفسیر سنتی ، JVM هر دستورالعمل bytecode را می خواند و آن را یک به یک اجرا می کند ، که می تواند نسبت به اجرای کد Native نسبتاً کندتر باشد.

کامپایل JIT این فرآیند را با ترجمه پویا بخش هایی از بایت کد به کد ماشین درست قبل از اجرای آنها بهینه می کند. کامپایلر JIT بخش هایی از کد را که مکررا اجرا می شوند(Hotspot)، شناسایی می کند و آنها را به یک کد بسیار بهینه سازی شده خاص برای سخت افزار و سیستم عامل کامپایل می کند. در اینجا نحوه کار کامپایل JIT به طور معمول آمده است:

1. Interpretation: در ابتدا، JVM بایت کد را تفسیر می کند و آن را دستور به دستور اجرا می کند.

2. Profiling: با تفسیر کد بایت، JVM اطلاعات مربوط به اجرای برنامه مانند فراوانی method invocations ها و یا تایپ آبجکت های استفاده شده در کد را جمع آوری می کند. عملیات Profiling به شناسایی نقاط HotSpot در کد کمک می کند.

3. Just-In-Time Compilation: پس از شناسایی Hotspotها، کامپایلر JIT شروع به کار می کند و کد مربوطه را به کد native به زبان ماشین ترجمه می کند.

4. Optimizations: در طول فرآیند کامپایل JIT، کامپایلر بهینه سازی های مختلفی را برای بهبود عملکرد روی بایت کد اعمال می کند. این بهینه سازی ها ممکن است شامل inlining method calls، حذف چک کردن های اضافی ، انجام constant folding و دستورالعمل هایی برای بهبود استفاده از Cache باشد.

5. Execution of Native Code: زمانی که بایت کد به زبان ماشین تبدیل می شود، با فراخوانی نقاط Hotspot مراحلی از کامپایل دیگر نیاز نخواهد بود که همین سبب بهبود پرفورمنس برنامه می شود.

کامپایل JIT هم مزایای استفاده از یک مفسر میانی و هم مزایای زبان های برنامه نویسی دیگر مانند C را (که بدون وجود مفسر میانی و مستقیما کد را به زبان ماشین تبدیل می کنند) ترکیب کرده و ارائه می دهد. این به JVM اجازه می دهد تا اجرای کد جاوا را به سرعت آغاز کند و در عین حال بهبودهای پرفورمنسی مشابه زبان های کامپایل شده سنتی را ارائه دهد. JVM به صورت پویا بخش های متداول کد را شناسایی و بهینه می کند و در نتیجه عملکرد کلی بهتر در طول زمان انجام می شود.

**الناز رفیعی پور**