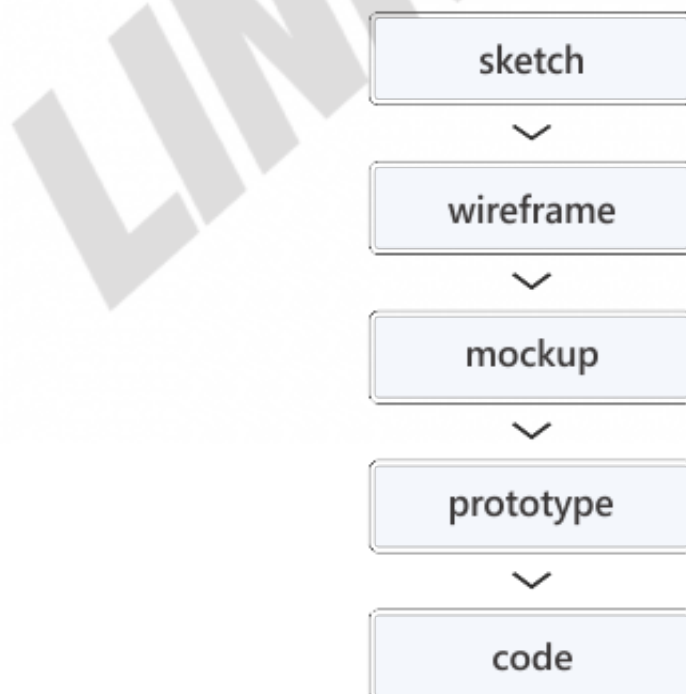


Din lecțiile anterioare ați putut presupune că procesul de creare a unui program poate fi mai mult decât complex. Pe lângă diferite limbaje pentru crearea programelor, la o astfel de operațiune pot participa multe instrumente specializate. Utilizarea instrumentului depinde, în primul rând, de tipul de program creat și de caracteristicile acestuia. În această lecție veți avea ocazia să vă familiarizați cu cele mai importante instrumente care sunt folosite într-un astfel de proces.

Instrumentele pentru proiectarea interfeței cu utilizatorul (UI)

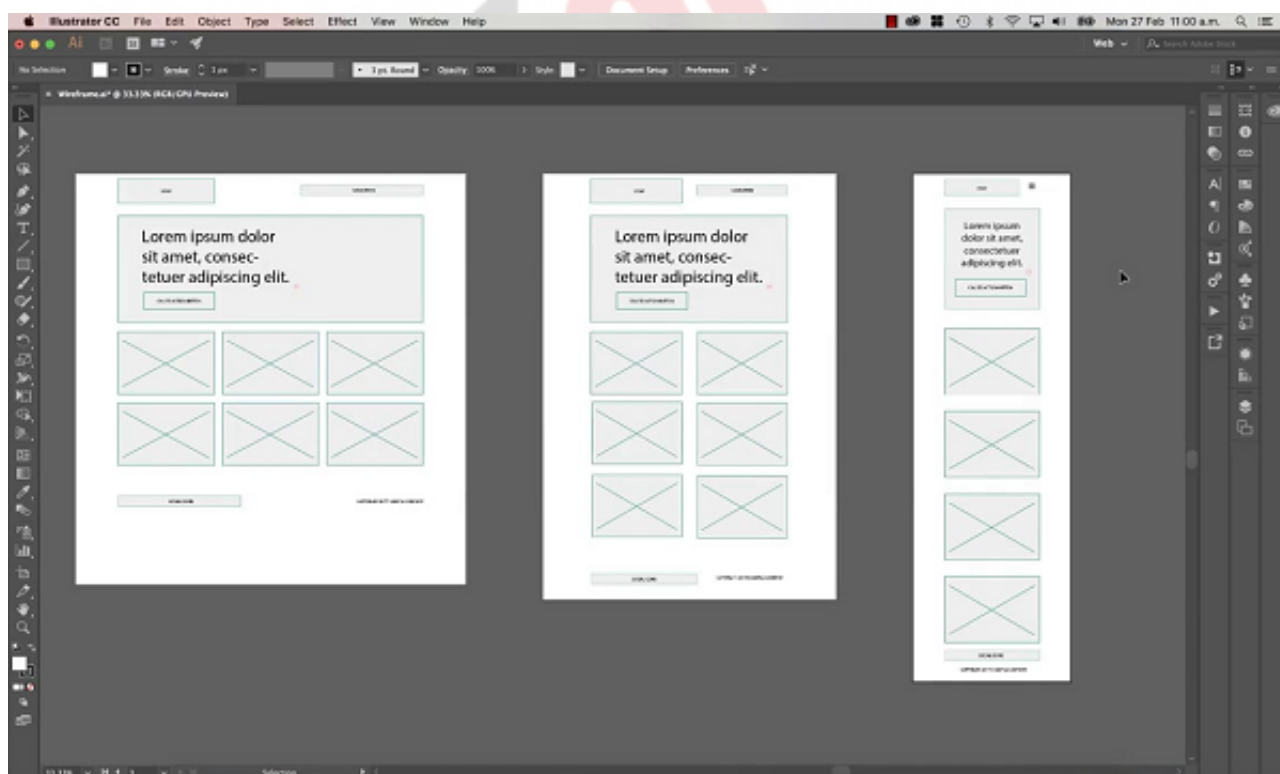
Dacă dezvoltați un program de calculator care are un mediu de utilizator grafic, înainte de implementarea concretă a aplicației, echipa de proiectare va dori cu siguranță să creeze o schiță de proiectare aproximativă sau detaliată. De fapt, procesul de realizare a mediului grafic este, în general, precedată de mai multe etape diferite (imaginea 3.1.).



Imaginea 3.1. Etapele realizării mediului de utilizator al unei aplicații

Imaginea 3.1. ilustrează toate etapele relevante care pot fi parcurse la proiectarea mediului de utilizator al unui program. Se începe cu desenarea unei schițe și se continuă prin crearea unor schițe aproximative, care sunt reprezentate de [wireframe](#)-uri. O schiță cu puțin mai multe detalii se numește mockup. După crearea mockup-ului, poate urma crearea reprezentărilor interactive - așa-numitelor prototipuri.

Wireframe-urile și mockup-urile pot fi create folosind programe de grafică de uz general, cum ar fi Adobe Photoshop și Illustrator. Illustrator este cel mai adesea folosit pentru crearea wireframe-urilor (imaginea 3.2.).



Imaginea 3.2. Exemplu de creare a wireframe-ului în Adobe Illustrator

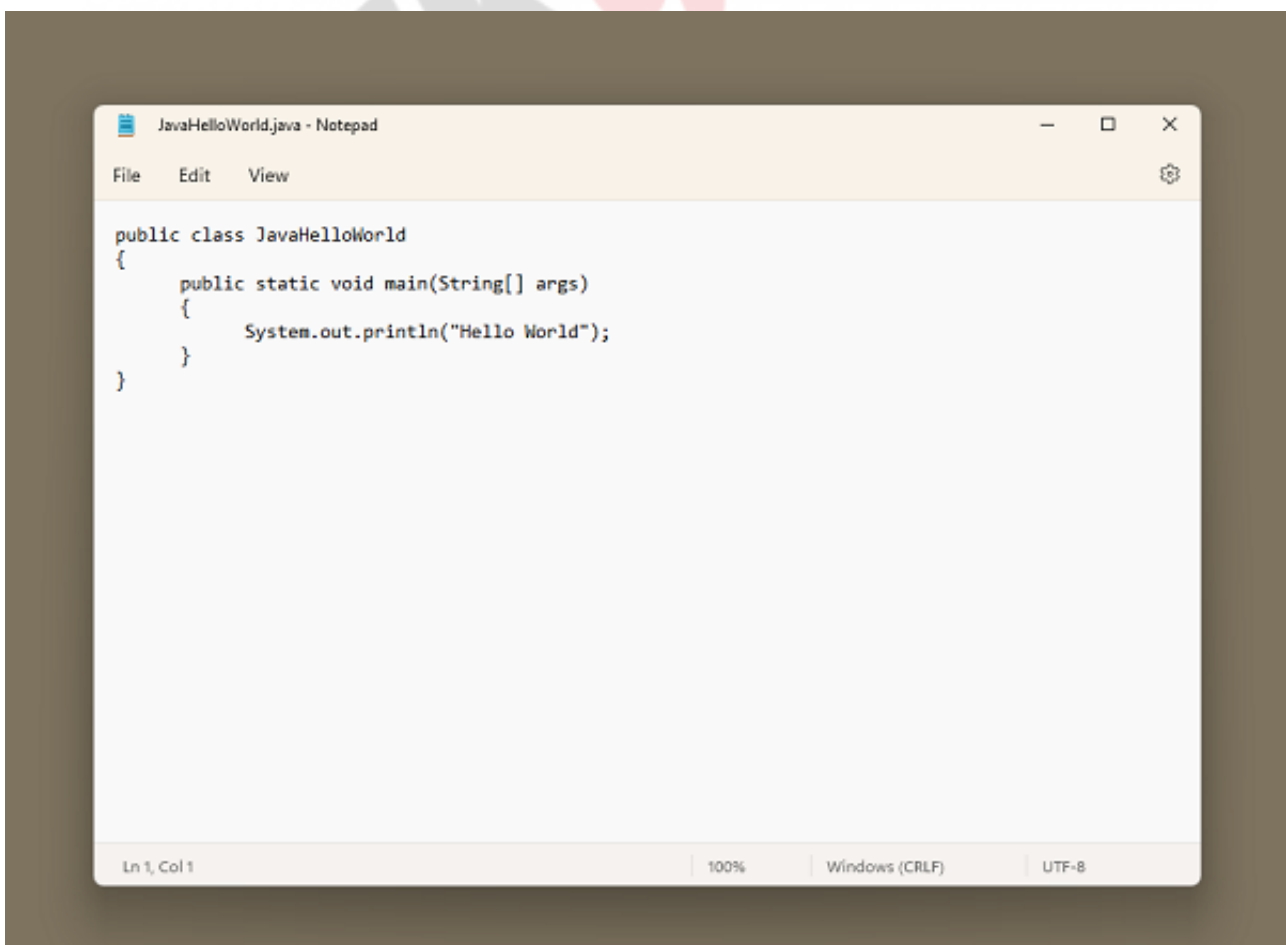
Modelele interactive sau prototipurile pot fi create folosind o combinație de Adobe Photoshop și [Adobe XD](#). Pe lângă programele de

mai sus, există și un număr mare de instrumente online care pot fi folosite pentru a crea wireframe-uri, mockup-uri și prototipuri - Miro, Figma, Sketch, Zeplin etc.

Scrierea codului sursă

Scrierea codului sursă este primul lucru care ne vine în minte atunci când se menționează profesia de programator. Și aceasta este într-adevăr o realitate - un programator își petrece cea mai mare parte a timpului său de lucru scriind codul sursă. Codul sursă este, de fapt, un text simplu care a fost formulat într-un mod special, astfel încât să satisfacă regulile de sintaxă ale limbajului folosit.

Nu este nevoie de un instrument special pentru a scrie codul sursă. Orice editor de text disponibil pe sistemul de operare al computerului este suficient (imaginea 3.3.).

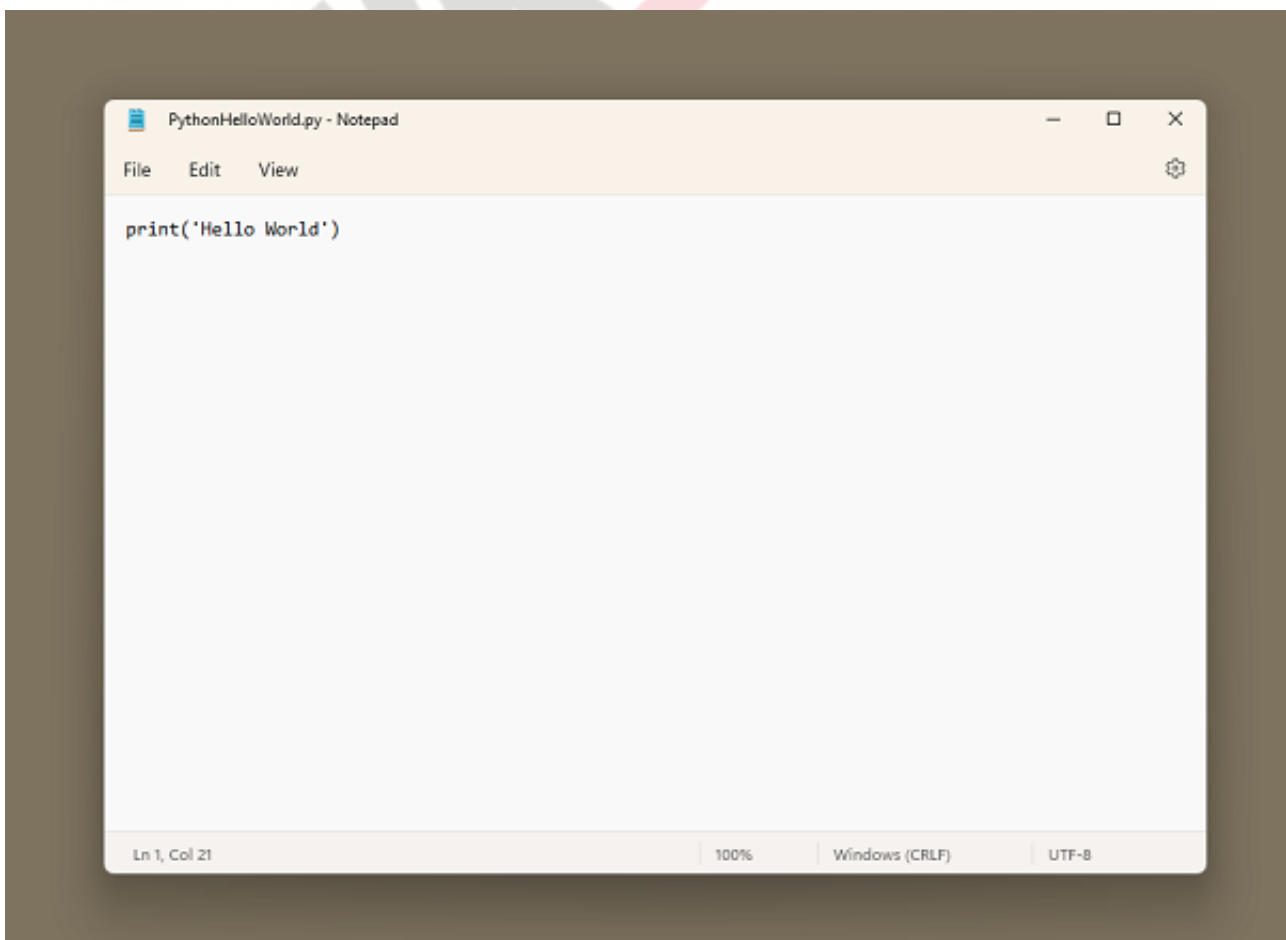


Imaginea 3.3. Programul Hello World scris cu limbajul de programare Java

În imaginea 3.3. puteți vedea programul Notepad pe sistemul de operare Windows, în care este scris un program complet funcțional. Este vorba de un program scris în limbajul Java, care la ieșire emite mesajul *Hello World*.

Codul sursă Java este un text special formulat care, după ce a fost compilat în codul mașină, poate trimite instrucțiuni specifice unui computer. Pe exemplul primului nostru program, o astfel de instrucțiune este să listăm mesajul *Hello World*.

Un lucru identic poate fi realizat folosind limbajul de programare Python, așa cum este ilustrat în imaginea 3.4.



Imaginea 3.4. Programul Hello World scris cu limbajul de programare Python

După ce am scris primul cod sursă al unui program simplu, întrebarea logică este cum să folosim un astfel de cod sursă pentru a trimite comenzi către computer. Răspunsul se află în povestea din prima lecție a acestui curs. Apoi am spus că codul sursă al unui limbaj de nivel superior a fost compilat în codul mașină, astfel încât un computer să-l poată executa. Prin urmare, următorul grup de instrumente de dezvoltare care va fi abordat în această lecție este tocmai cel care permite compilarea codului sursă.

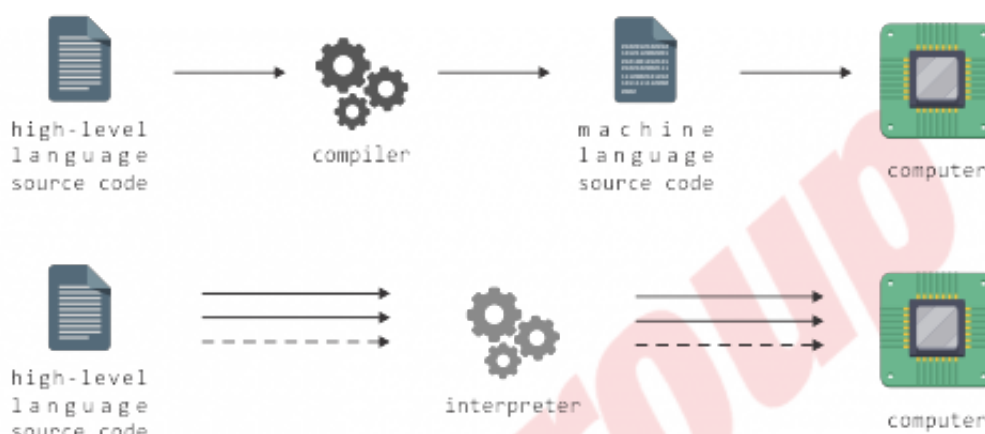
Instrumentele pentru compilarea codului sursă

Compilarea codului sursă se poate face în două moduri de bază – prin compilare și interpretare.

Compilatorul (*compiler* în engleză) este un program de calculator al cărui scop principal este generarea altor programe de calculator. Un compilator compilează/traduce codul sursă al programelor scrise în limbaje de nivel superior în codul mașină, astfel încât un cod de genul acesta să poată fi executat de un computer. Operațiunea pe care o execută compilatorul se numește compilare, iar limbajele care folosesc compilatoare sunt limbaje compilate. Operațiunea de compilare se face dintr-o dată, înainte ca un anumit program scris într-un limbaj superior să fie executat pe un computer. Rezultatul compilării este un fișier executabil cu un conținut care constituie codul mașină.

Interpretorul face aceeași treabă ca un compilator, dar într-un mod ușor diferit. Spre deosebire de compilator, interpretorul compilează părți mai mici ale codului sursă în timpul execuției acestuia. Astfel, prin folosirea interpretorului, nu este necesară compilarea întregului program, ci aceasta se face pe părți, în timpul execuției.

Diferențele dintre compilare și interpretare sunt ilustrate în imaginea 3.5.



Imaginea 3.5. Diferențe între modurile de operare ale compilatorului și interpretorului

După cum puteți vedea, procesul de compilare va genera un fișier complet nou care conține codul mașină complet al programului compilat. Odată compilat, programul poate fi executat de un număr arbitrar de ori pe diferite computere. Pe de altă parte, interpretarea se face în mai multe etape, instrucțiune cu instrucțiune, astfel încât codul mașină este generat din mai multe părți, care sunt executate de calculator imediat după compilare.

Procesul de interpretare asigură o rulare inițială mai rapidă a programului, deoarece întregul cod sursă nu este compilat deodată. Pe de altă parte, odată ce procesul de compilare a codului sursă al limbajului de nivel superior în limbajul mașină este încheiat, fiecare rulare și execuție ulterioară este semnificativ mai rapidă, având în vedere că programul există deja sub formă de mașină.

Compilarea are un alt avantaj față de interpretare. Deoarece codul sursă complet este compilat înainte de execuție, compilatorul are capacitatea de a verifica codul sursă scris. Dacă codul sursă conține o

eroare de sintaxă, compilatorul nu va putea compila un astfel de program și, prin urmare, programul nu va fi executat. În acest fel, este împiedicată executarea programelor care au o eroare în codul sursă. Pe de altă parte, detectarea erorilor de sintaxă nu este posibilă în timpul procesului de interpretare, dat fiind că interpretorul compilează codul în mai multe părți, instrucțiune cu instrucțiune, în timpul execuției acestuia. Dacă codul sursă interpretat conține o eroare, aceasta va apărea cu siguranță în timpul execuției programului.

Limbajul de programare este doar o specificație

Limbajele de programare moderne cu greu pot fi clasificate în unele compilate și interpretate. Având în vedere că un anumit limbaj de programare este doar o specificație, procesul specific de implementare a limbajului determină modul în care va fi compilat în limbajul mașină. Acest lucru înseamnă practic că aproape orice limbaj poate fi interpretat și compilat și că metoda specifică de compilare depinde de cine creează implementarea limbajului specific. Un limbaj de programare nu este altceva decât un set de anumite reguli care sunt convenite în prealabil. Un astfel de set de reguli determină sintaxa limbajului, care definește în cele din urmă aspectul și structura codului pe care programatorii îl vor scrie. Pentru ca un limbaj de programare de nivel superior să fie cu adevărat funcțional, este necesară scrierea unei componente de programare care să compileze codul unui astfel de limbaj în limbajul mașină. La implementarea unei astfel de componente, trebuie să se implementeze în mod concret toate acele reguli de limbaj care au fost agreeate anterior prin specificație, completată cu limbajul de programare.

Limbajele de programare moderne de nivel superior sunt adesea compilate prin combinarea compilării și interpretării. Implementările unor limbaje, pe de altă parte, se bazează exclusiv pe una dintre cele două metode menționate de compilare a codului sursă. De exemplu, limbajul de programare C a fost conceput în primul rând pentru a fi compilat, care este cel mai frecvent caz în implementările sale. Aceeași situație există și cu limbajele de programare Objective

C și Swift, limbajele folosite pentru crearea programelor pentru dispozitivele Apple. Pe de altă parte, un limbaj de programare la nivel superior foarte popular numit JavaScript este un exemplu de limbaj de programare interpretat. Un exemplu de limbaj interpretat este Python, al cărui cod sursă l-ați putut vedea mai devreme. Limbajele precum C#, Java și PHP sunt în primul rând compilate, dar procesul lor de compilare include și unele caracteristici de interpretare.

Mai devreme în această lecție, a fost prezentat codul sursă al unui program (Hello World) scris folosind două limbaje de programare diferite – Java și Python. Java este un limbaj de programare compilat primar, în timp ce Python este un exemplu clasic de limbaj interpretat. În final, asta înseamnă că în rândurile următoare va trebui să folosim atât un compilator, cât și un interpretor pentru a compila programele pe care le-am scris într-o formă de mașină, adică o formă care este potrivită pentru execuție de către hardware-ul computerului.

Interpretorii și compilatoarele necesare pentru a compila codul aproape tuturor limbajelor de programare de uz general sunt disponibile ca parte integrantă a setului de instrumente mai mari, disponibile de la companiile care au creat limbajul de programare specific. Astfel de seturi pot fi găsite pe site-urile oficiale în mai multe versiuni diferite, în funcție de sistemul de operare pe care îl utilizați. În continuare vom ilustra separat descărcarea lor pentru Java și Python și apoi prezentăm și utilizarea lor pentru compilarea și rularea codului sursă pe care l-am scris.

Compilarea și rularea codului Java

Pentru început, ne vom ocupa de configurarea mediului pentru executarea codului Java. Un astfel de mediu este unificat într-un set de instrumente numit *Java Developer Kit*, pe scurt JDK. În continuare vom descărca și configura mai întâi JDK. După aceea vom ilustra procesul de compilare și rulare a unui program Java.

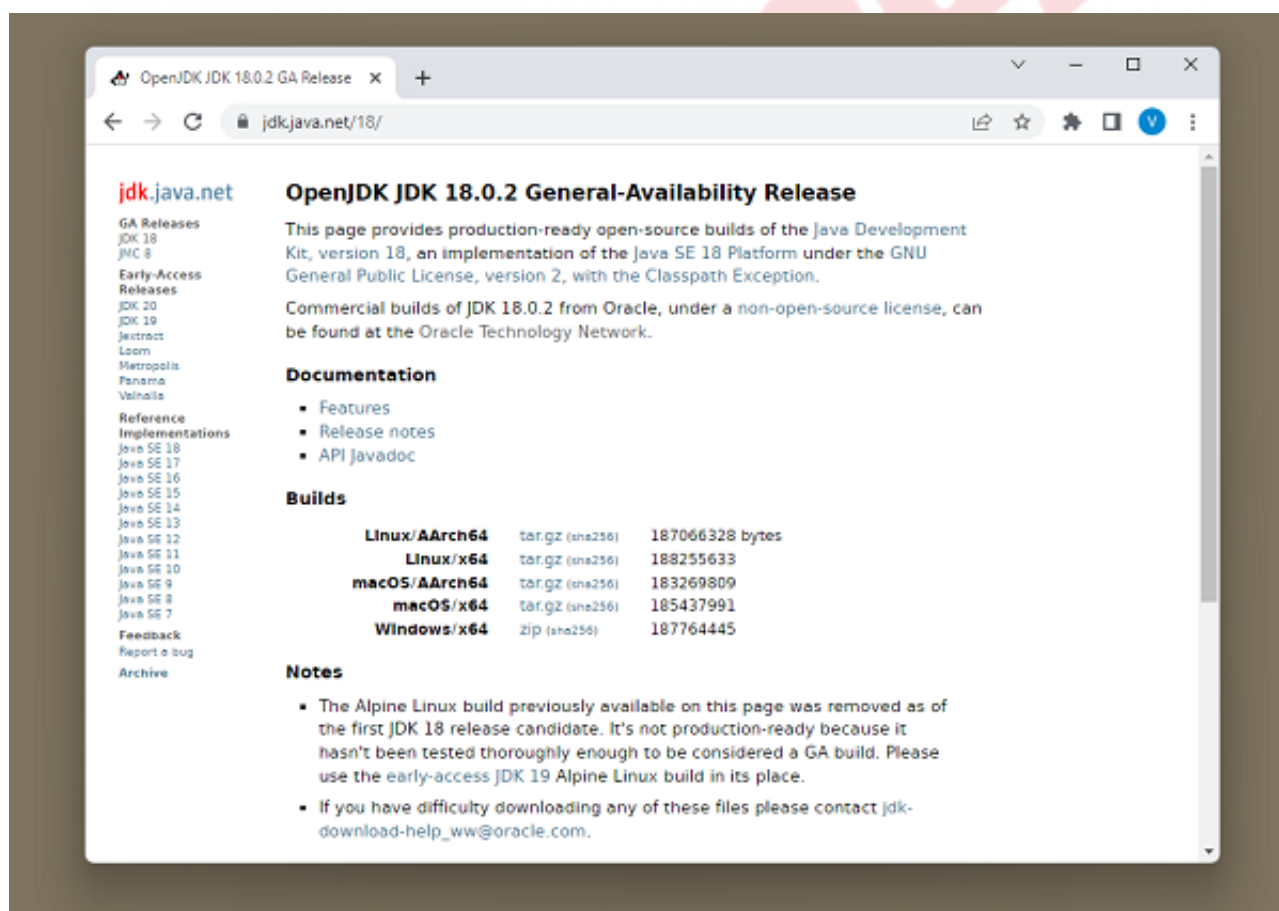
Descărcarea și configurarea Java Developer Kit

În acest curs vom folosi **OpenJDK**, un JDK cu sursă deschisă complet

gratuit de la Oracle. OpenJDK poate fi descărcat de la următoarea adresă:

<https://jdk.java.net/18/>

Pagina de la adresa afișată are linkuri pentru descărcarea variantei OpenJDK care corespunde sistemului de operare al computerului vostru (imaginea 3.6.).



Imaginea 3.6. Pagina web pentru descărcarea OpenJDK-ului

În secțiunea **Builds** puteți vedea linkuri pentru descărcarea versiunii corespunzătoare a OpenJDK. Dând clic pe unul dintre linkuri, începe descărcarea fișierelor OpenJDK.

După ce descărcarea OpenJDK este completă, trebuie să dezarhivați arhiva descărcată. Se recomandă să creați un folder cu numele Java pe partiția primară a hard diskului și să despachetați conținutul arhivei descărcate în acesta. În acest fel, veți obține următorul folder pe computer:

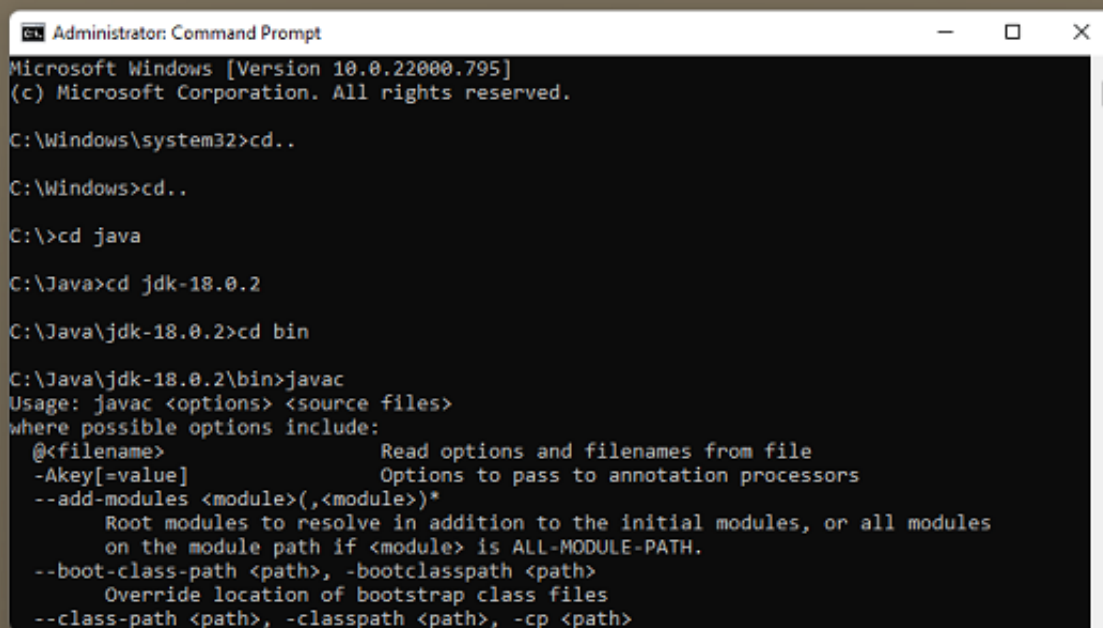
```
C:\Java\jdk-18.0.2
```

La calea specificată vor exista mai multe foldere, dintre care folderul **bin** este cel mai important pentru noi în acest moment:

```
C:\Java\jdk-18.0.2\bin
```

Folderul `bin` conține toate instrumentele necesare pentru pornirea și dezvoltarea programelor Java, inclusiv compilatorul Java (este vorba de fișierul numit `javac`).

Compilatorul Java tocmai descărcat se poate utiliza în modul ilustrat în imaginea 3.7.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.22000.795]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd..

C:\Windows>cd..

C:\>cd java

C:\Java>cd jdk-18.0.2

C:\Java\jdk-18.0.2>cd bin

C:\Java\jdk-18.0.2\bin>javac
Usage: javac <options> <source files>
where possible options include:
  @<filename>           Read options and filenames from file
  -Akey[=value]         Options to pass to annotation processors
  --add-modules <module>(<module>)*
                        Root modules to resolve in addition to the initial modules, or all modules
                        on the module path if <module> is ALL-MODULE-PATH.
  --boot-class-path <path>, -bootclasspath <path>
                        Override location of bootstrap class files
  --class-path <path>, -classpath <path>, -cp <path>
```

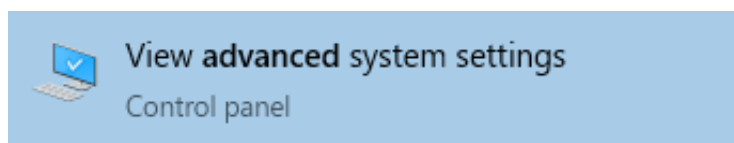
Imaginea 3.7. Exemplu de utilizare a compilatorului Java

În imaginea 3.7. puteți vedea cum, folosind consola sau terminalul, ne-am poziționat mai întâi în folderul `bin` al JDK-ului și apoi am apelat **`javac`**, respectiv compilatorul Java. Pentru a nu fi nevoiți să ne poziționăm de fiecare dată în folderul `bin`, cel mai bine este să plasăm locația în care se află compilatorul Java în variabila de [sistem PATH](#).

Adăugarea căii la variabila de sistem Path

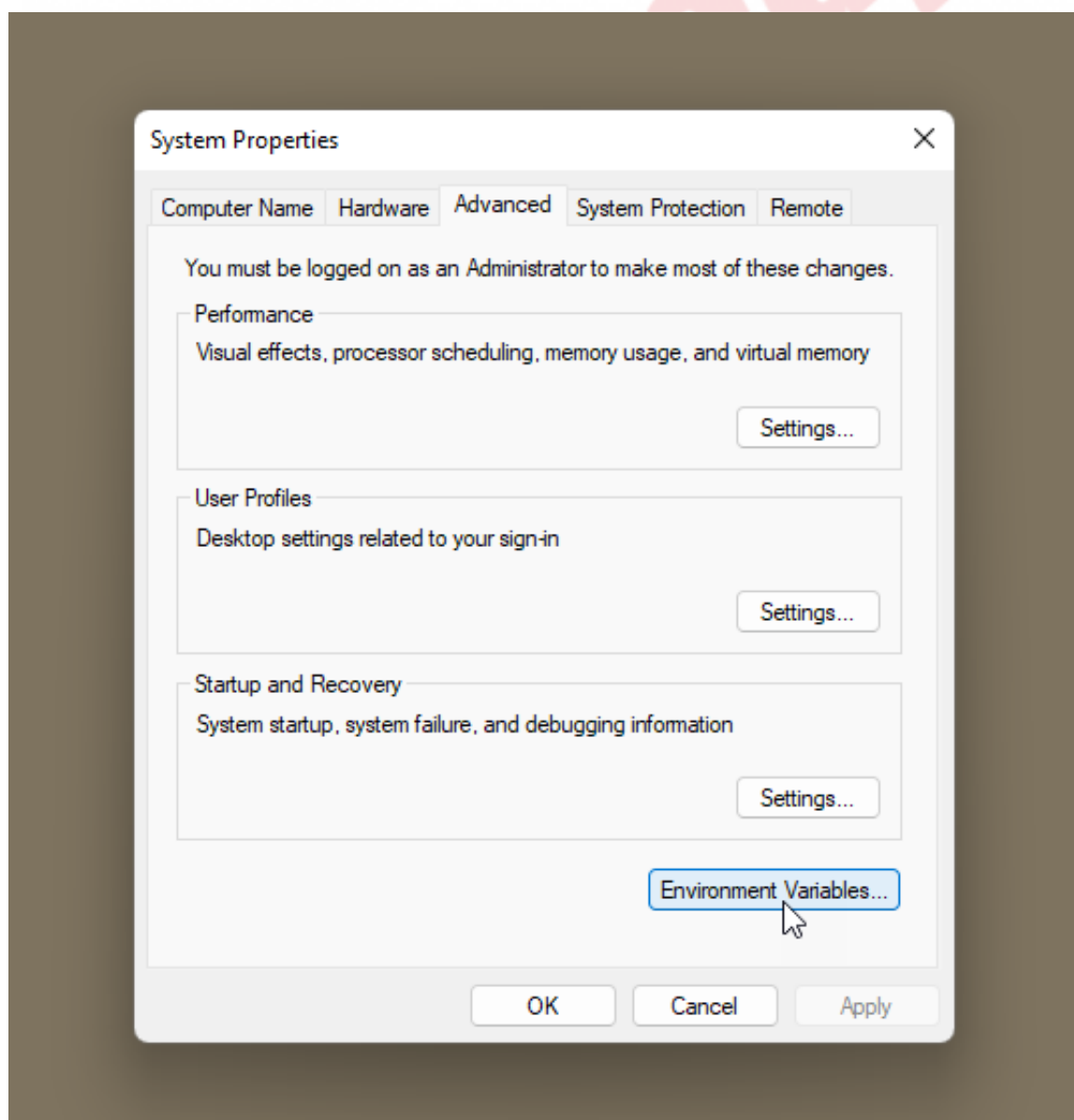
Calea la care se află folderul `bin` este adăugată la variabila de sistem Path în felul următor (sistemele de operare Windows 10 și Windows 11).

1. Deschideți fereastra *Advanced System Settings* (imaginea 3.8.).



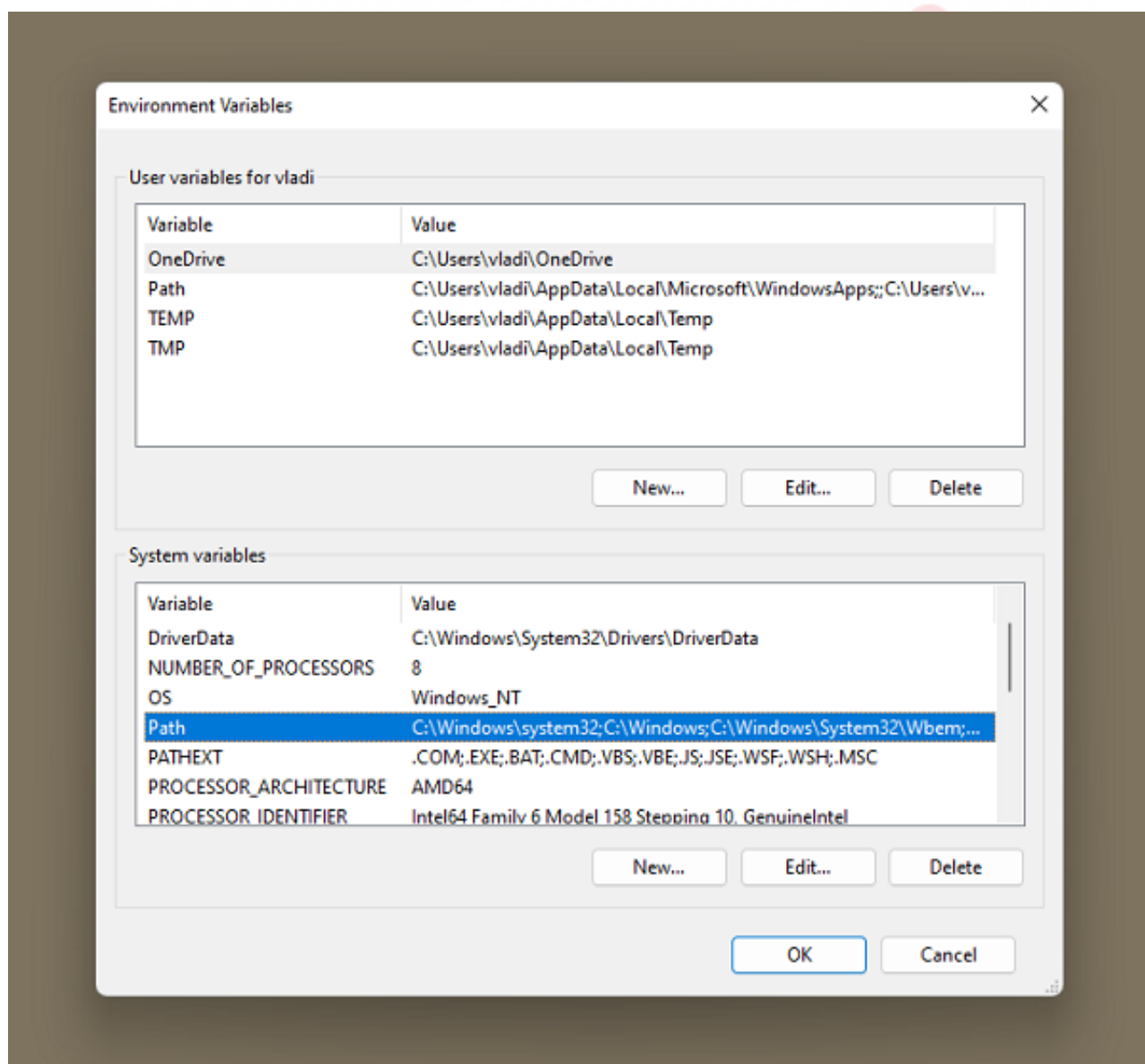
Imaginea 3.8. Opțiunea pentru deschiderea setărilor de sistem avansate

2. În partea de jos a ferestrei alegeți opțiunea *Environment Variables* (imaginea 3.9.).



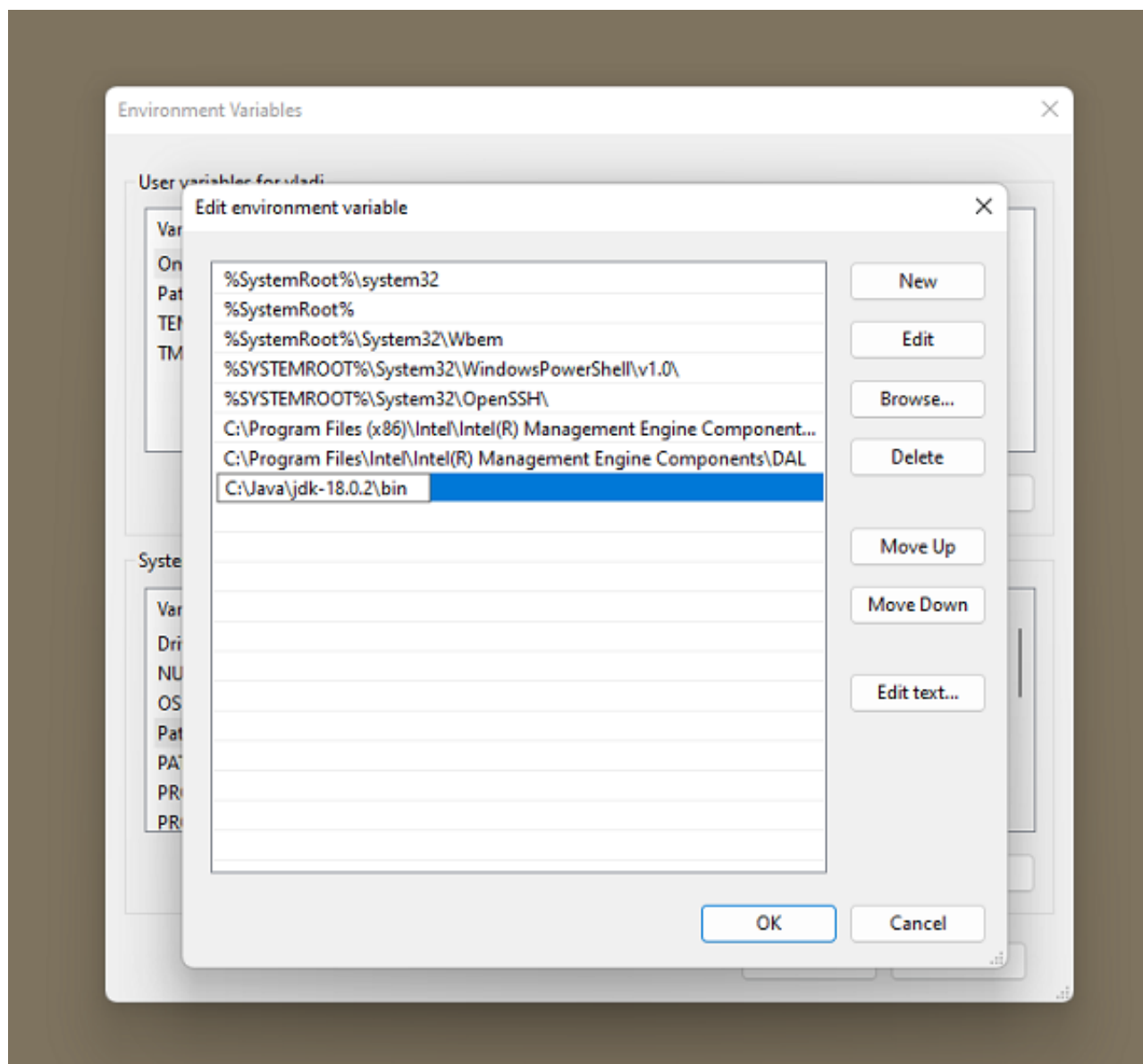
Imaginea 3.9. Fereastra cu setări de sistem avansate

3. În secțiunea *System variables* găsiți variabila *Path* și dați dublu clic (imaginea 3.10.).



Imaginea 3.10. Variabila de sistem Path

4. În fereastra obținută adăugați o nouă cale la care se găsește folderul `bin` al pachetului JDK (imaginea 3.11.).

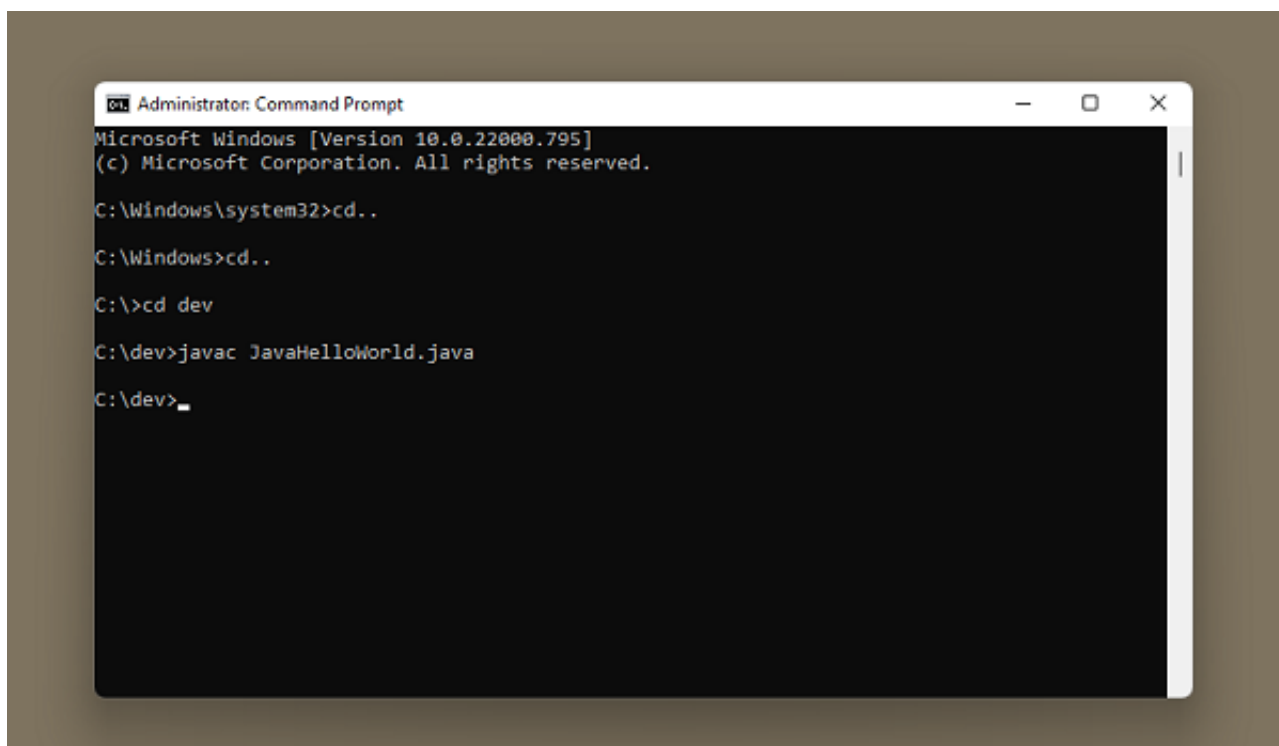


Imaginea 3.11. Adăugarea unei noi înregistrări în variabila Path

5. Confirmați totul cu clic pe butonul OK.

După setarea locației în care se află instrumentele Java în variabila PATH, aceste instrumente pot fi folosite mult mai simplu tastând `java` sau `javac`, indiferent de folderul în care ne aflăm în prezent. De asemenea, va fi mult mai ușoară compilarea codului Java

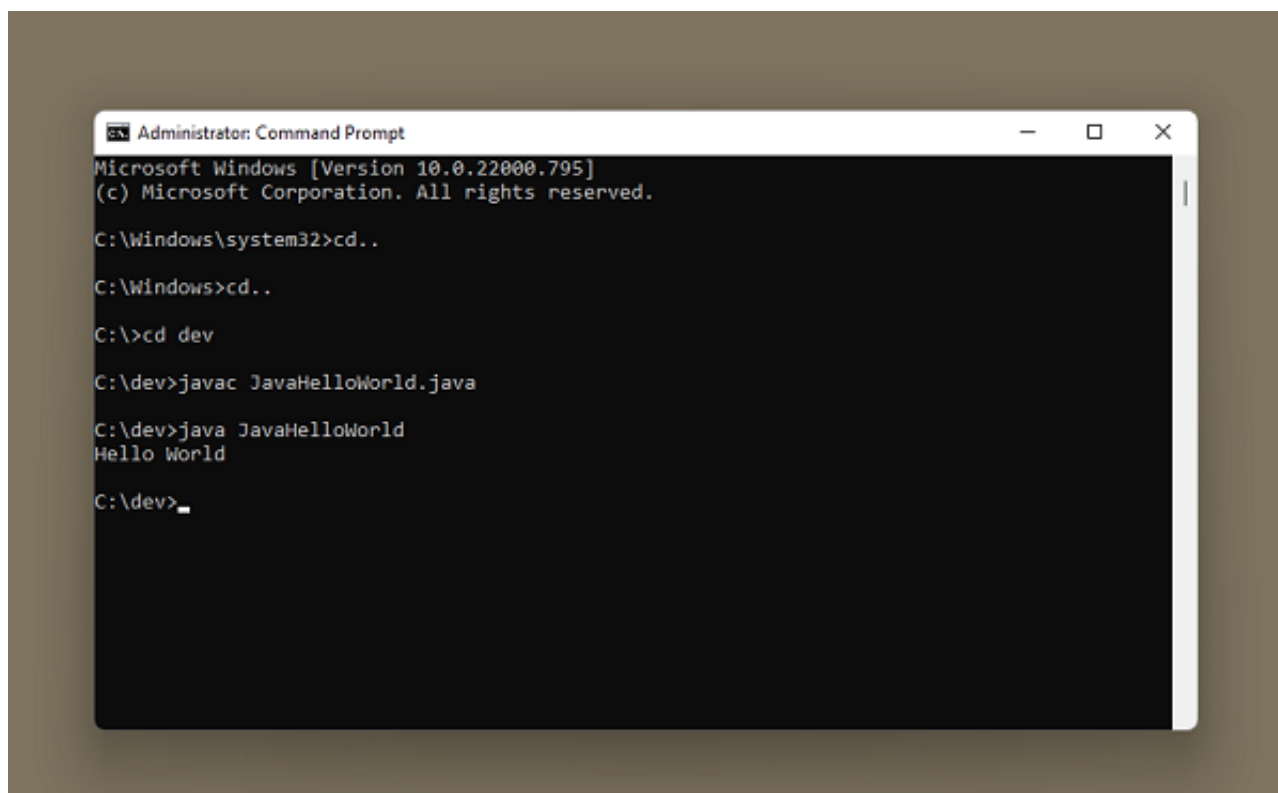
sursă pe care l-am scris la începutul acestei lecții (imaginea 3.12.).



Slika 3.12. Compilarea programului Java

În imaginea 3.12. puteți vedea cum este compilat codul sursă Java, mult mai simplu de data aceasta, deoarece instrumentele Java sunt plasate în variabila PATH.

Finalizarea cu succes a compilării unui program Java înseamnă crearea unui alt fișier în folderul cod sursă. Este vorba de un fișier care are același nume ca cel cu codul sursă, dar cu extensia `.class`. Conținutul unui astfel de fișier este [bytecode](#), pe care mediul Java îl poate rula direct (imaginea 3.13.).



Imaginea 3.13. Compilarea programului Java

În imaginea 3.13. puteți vedea modul în care programul este pornit și efectul execuției acestuia. Programul este rulat folosind instrumentul **Java**. Efectul execuției programului este listarea mesajului Hello World la ieșire.

Interpretarea și rularea codului Python

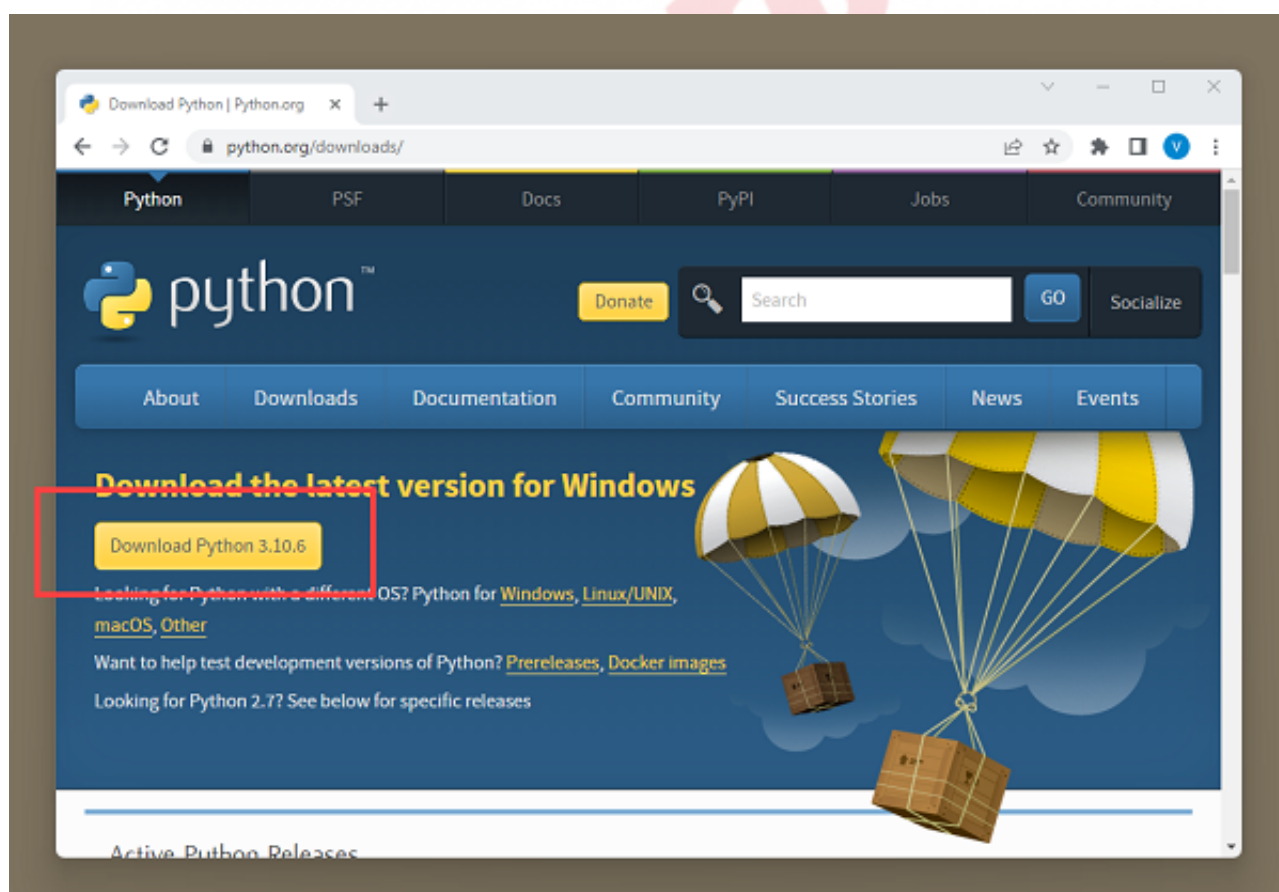
Compilarea codului Python pe care l-am scris mai devreme se face într-un mod similar cu compilarea codului Java din rândurile anterioare. În continuare vom ilustra mai întâi descărcarea și configurarea mediului Python, apoi compilarea și rularea programului *Hello World*.

Descărcarea și configurarea mediului Python

Python se poate descărca de pe site-ul oficial:

<https://www.python.org/downloads/>

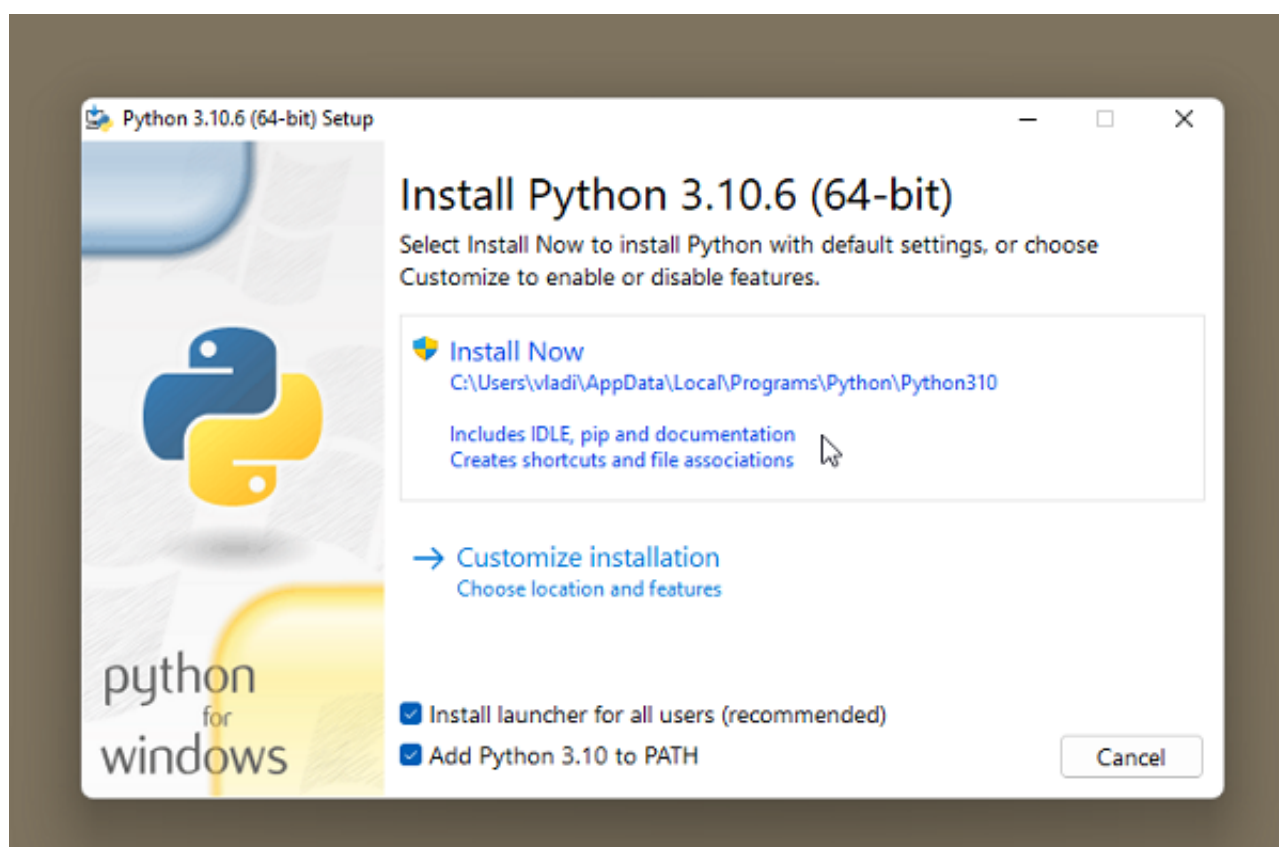
Pe site-ul specificat puteți să găsiți o mulțime de forme diferite în care puteți să descărcați Python. Predomină fișierele de instalare care automatizează întregul proces de configurare a unui mediu pentru execuția programelor Python. Noi vom opta pentru o astfel de abordare, adică să descărcăm fișierul de instalare care va face totul pentru noi (imaginea 3.14.).



Imaginea 3.14. Descărcarea Python-ului

În imaginea 3.14. puteți vedea că alegem butonul *Download Python 3.10.6*. După descărcarea fișierului de instalare, acesta trebuie

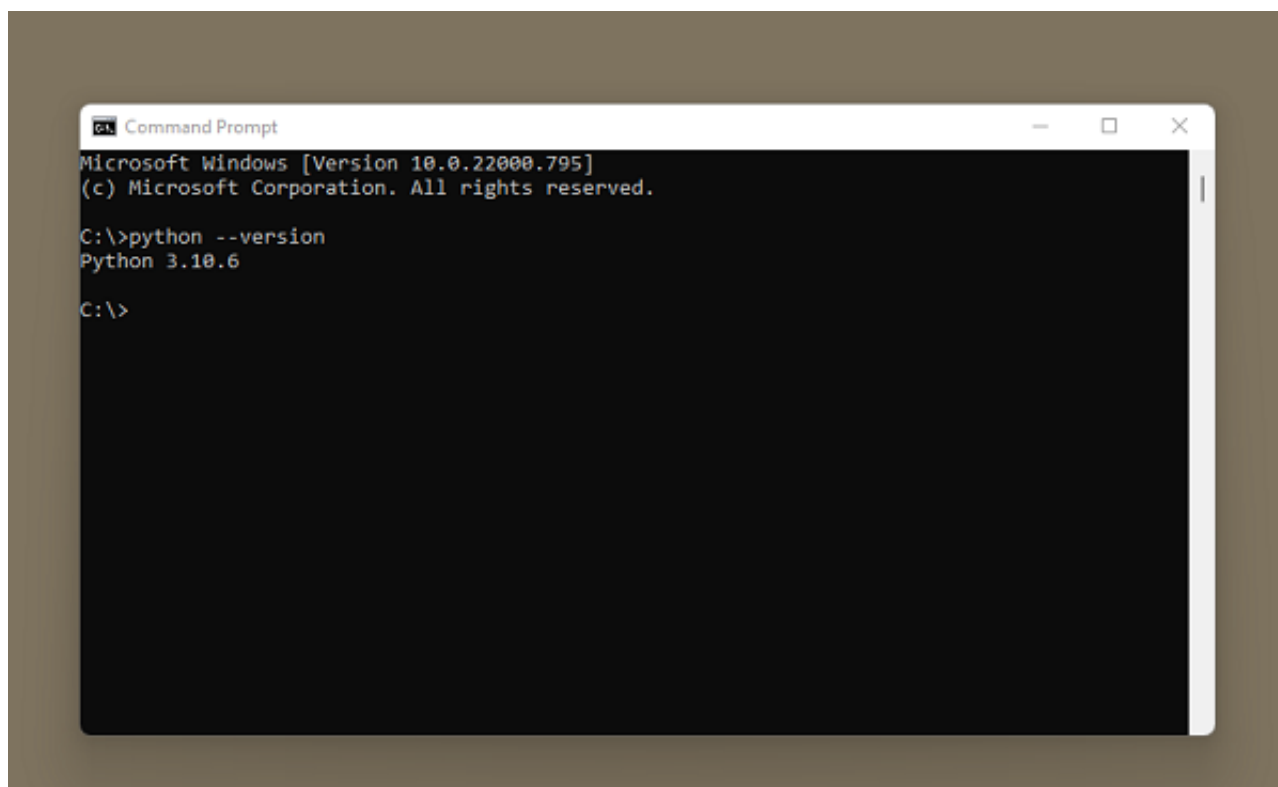
rulat (imaginea 3.15.).



Imaginea 3.15. Instalarea Python-ului

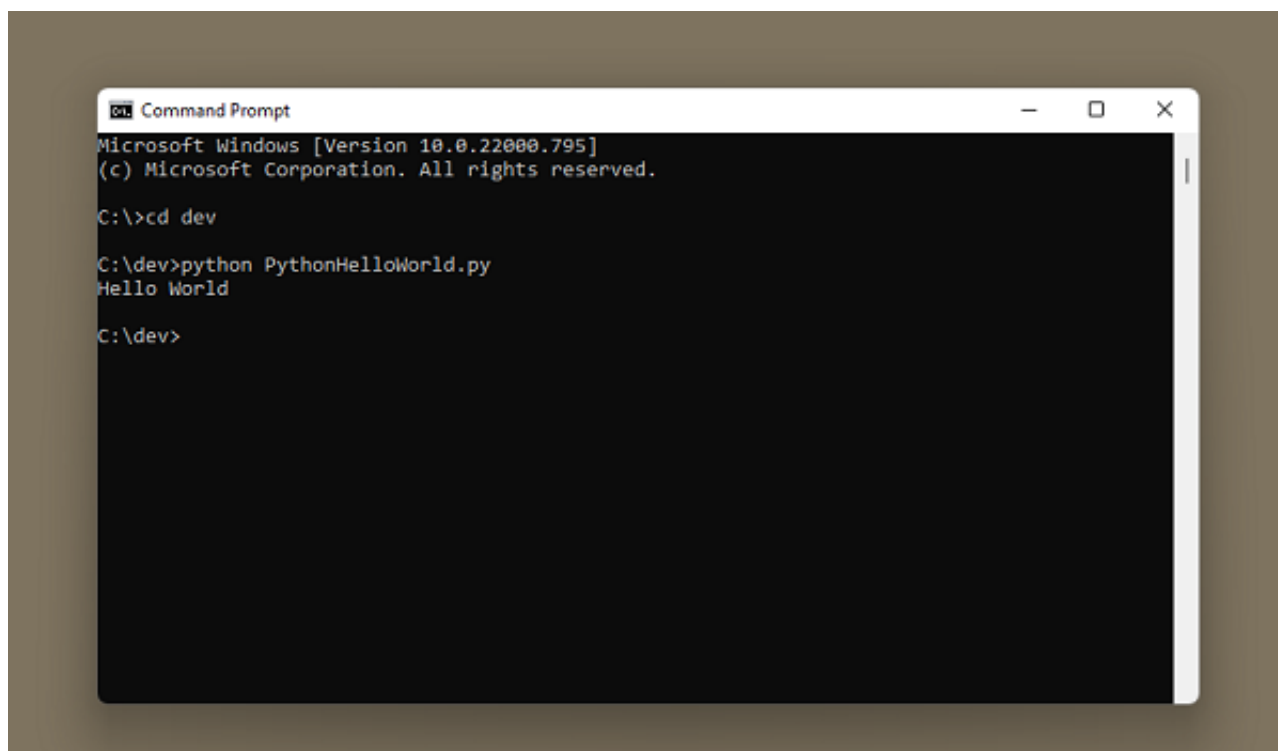
În imaginea 3.15. puteți vedea fereastra care se deschide după pornirea instalării Python. Este important de observat ultima opțiune care este activată - *Add Python 3.10 to PATH*. Aceasta este o opțiune care va avea ca rezultat plasarea căii unde va fi instalat Python în variabila de sistem PATH. Cu alte cuvinte, ceea ce am făcut independent pentru instrumentele Java se va face automat.

După finalizarea instalării Python, va fi posibilă activarea directă a interpretorului Python, prin specificarea comenzii **python**. De exemplu, pentru a verifica versiunea disponibilă de Python, este suficient să introducem comanda `python --version`, ca în imaginea 3.16.



Imaginea 3.16. Determinarea versiunii de Python

După stabilirea mediului Python, codul sursă pe care l-am scris mai devreme poate fi compilat și rulat cu ușurință, așa cum se poate vedea în imaginea 3.17.



Imaginea 3.17. Compilarea și rularea programului Python

Editoarele de text

Rândurile anterioare au avut ca scop să vă ilustreze cel mai scăzut nivel posibil pentru scrierea și compilarea programelor. Cu toate acestea, în practică, pentru dezvoltarea software sunt utilizate în general diverse programe specializate, care permit o muncă mai rapidă și mai confortabilă. Unul dintre cele mai importante grupuri de astfel de programe sunt editoarele de text.

În forma lor cea mai de bază, editoarele de text sunt programe de calculator concepute pentru a procesa text simplu (*plain text* în engleză). Când spunem simplu, ne referim la text curat, fără [metadate](#) suplimentare legate de formatare, stil și alte caracteristici însoțitoare. Deși editoarele de text sunt practic universale, astăzi acest termen este folosit în principal pentru a reprezenta programe care sunt destinate dezvoltării software.

Primul și scopul de bază al editorului de text este capacitatea de a vizualiza, modifica și crea fișiere text. În plus, diferite editoare de text au un set diferit de funcționalități. Setul de bază de funcționalități pe care le are fiecare editor de text include:

- crearea, modificarea și revizuirea fișierelor text,
- copierea, mutarea și căutarea textului,
- suport pentru un număr mare de limbaje de programare, ceea ce înseamnă setarea automată a extensiei corespunzătoare la crearea fișierelor.

Pe lângă aceste caracteristici de bază, majoritatea editoarelor de text au următoarele funcționalități foarte utile:

- marcarea elementelor de limbaj, care în engleză se numește funcționalitate *syntax highlighting* și implică marcarea diferitor părți ale codului programului cu culori diferite, ceea ce îmbunătățește transparența;
- formatarea automată a codului computerului, care include indentarea liniilor și trecerile la un rând nou, din nou pentru îmbunătățirea vizibilității;
- abilități de navigare, care includ trecerea ușoară de la o secțiune la alta a codului computerului, precum și navigarea ușoară prin diferite fișiere ale aceluiași proiect;
- posibilitatea de upgrade - majoritatea editorilor de text moderni au un sistem de upgrade care permite extinderea setului inițial de funcționalități folosind diverse opțiuni de plugin sau add-on-uri.

Unele editoare de text populare au funcționalități și mai avansate, cum ar fi:

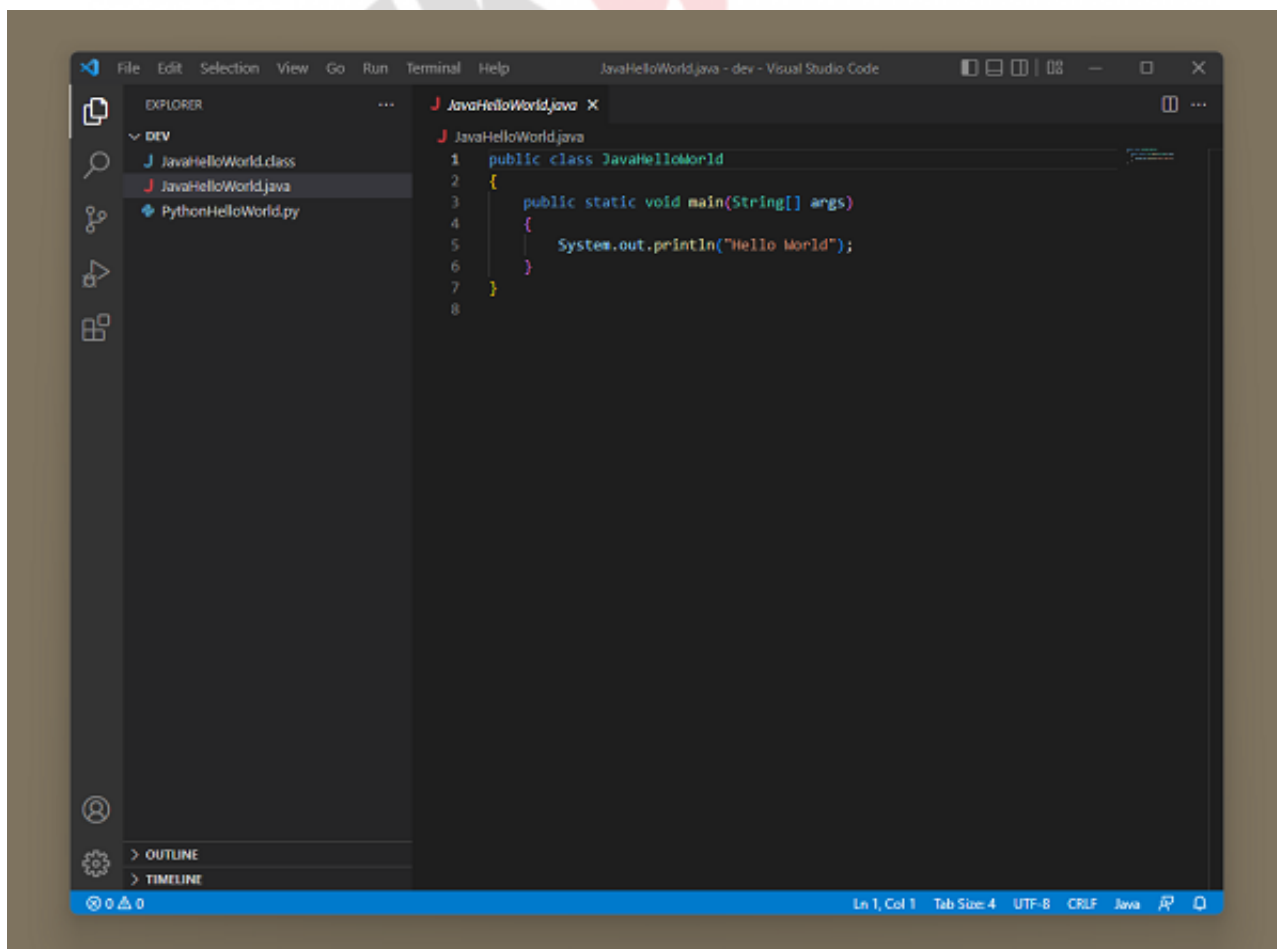
- completarea automată a codului,
- generarea codului,

- instrumente de testare și depanare.

Cele mai populare editoare de text sunt următoarele:

- Notepad++,
- Sublime Text,
- Atom,
- Visual Studio Code,
- Brackets.

Imaginea 3.18. ilustrează fereastra editorului Visual Studio Code în care este scris codul programului deja văzut.



Imaginea 3.18. Visual Studio Code

Mediile de dezvoltare

Pe lângă editoarele de text, pentru dezvoltarea software sunt folosite diverse programe, care sunt incluse în grupul de medii de dezvoltare integrate (*Integrated Development Environment*, prescurtat *IDE* în engleză). Mediile de dezvoltare integrate sunt programe de dezvoltare software care, de regulă, au și alte instrumente de dezvoltare în plus față de editoarele de text. Astfel de instrumente includ, în principal, instrumente pentru compilarea codului sursă în codul mașină, precum și diverse instrumente pentru testare și depanare. Realitatea este, totuși, că majoritatea editoarelor de text descrise în rândurile anterioare au unele caracteristici ale unui mediu de dezvoltare real, așa că se poate spune că linia care separă editoarele de text și mediile de dezvoltare de astăzi este foarte subțire.

Mediile de dezvoltare integrate au un set mai avansat de funcționalități, care includ în principal:

- **interpretorii și compilatoarele** - acestea sunt componente speciale ale mediilor de dezvoltare, care au scopul de a converti codul de programare al limbajelor de nivel superior în limbaj de mașină,
- **marcarea/evidențierea sintaxei** (syntax highlighting în engleză) - funcționalitate care afișează diferite părți ale codului într-un mod diferit, pentru a îmbunătăți vizibilitatea,
- **completarea automată a codului** (code completion în engleză) - prin începerea introducerii unui cuvânt-cheie al limbajului, mediul de dezvoltare completează în mod independent cuvântul început,
- **schimbarea automată a codului** (refactoring în engleză) - funcționalitate care permite scenarii avansate de modificare a codului programului,
- **controlul versiunilor** (version control în engleză) - funcționalitate care oferă versiunea codului în curs de scris,

astfel încât modificările să poată fi monitorizate și revenirea simplă la una dintre versiunile anterioare ale codului programului,

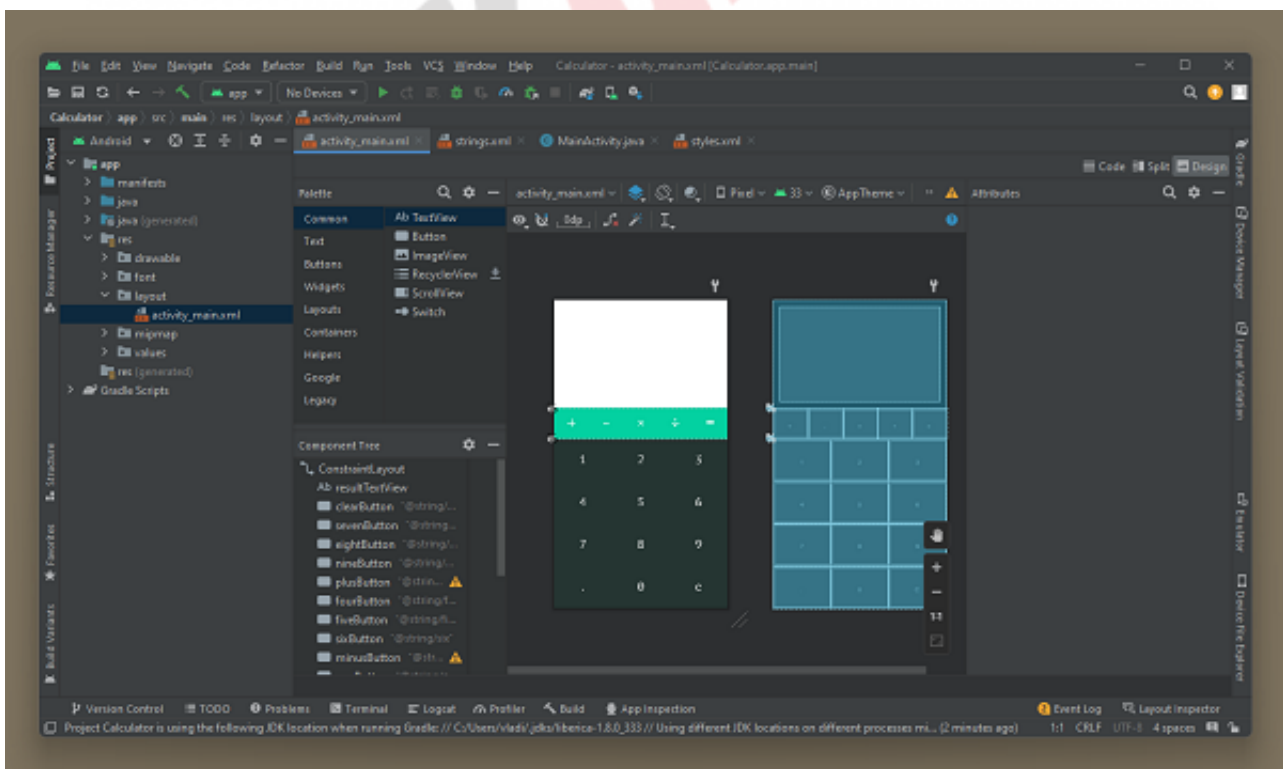
- **depanarea** (debugging în engleză) – un sistem care permite o depanare foarte ușoară prin monitorizarea execuției codului, căutarea codului,
- **suport pentru un număr mare de limbaje.**

În continuarea textului specificăm unele dintre cele mai cunoscute medii de dezvoltare pentru web.

- **Visual Studio** – mediul de dezvoltare al companiei Microsoft, care este destinat în primul rând dezvoltării aplicațiilor folosind limbaje și instrumente Microsoft; cu toate acestea, este vorba de un mediu de dezvoltare universal care poate fi folosit pentru a scrie codul de programare în aproape orice limbaj.
- **NetBeans** – mediu de dezvoltare destinat în primul rând dezvoltării folosind limbajul Java; cu toate acestea, de-a lungul timpului a fost adăugat și suport pentru limbajele PHP, C, C++, HTML și JavaScript; este vorba de un program disponibil pentru sistemele de operare Windows, macOS, Linux și Solaris.
- **Eclipse** – un alt mediu de dezvoltare care a câștigat popularitate pe baza suportului său pentru dezvoltarea aplicațiilor Java; pe lângă limbajul Java, are și suport pentru multe alte limbaje de programare.
- **IntelliJ IDEA** – mediul de dezvoltare JetBrains destinat creării programelor Java; există în două versiuni - Community, care se folosește gratuit, și Ultimate, care este un exemplu clasic de program trialware, adică un program care poate fi încercat pentru un anumit număr de zile, după care este necesară plata pentru utilizarea ulterioară a acestuia.
- **Android Studio** – mediu de dezvoltare al companiei Google, special destinat dezvoltării aplicațiilor Android; se bazează pe mediul de dezvoltare IntelliJ IDEA.
- **WebStorm** – mediu de dezvoltare JetBrains destinat în principal pentru crearea aplicațiilor JavaScript și programării web pe partea de client.

- **PhpStorm** – mediu de dezvoltare JetBrains pentru programarea PHP.
- **PyCharm** – mediul de dezvoltare JetBrains pentru crearea programelor folosind limbajul Python.
- **Xcode** – mediu de dezvoltare Apple destinat dezvoltării programelor pentru sistemele de operare macOS, iOS, iPadOS, watchOS și tvOS; este un mediu de dezvoltare gratuit care funcționează numai pe sistemul de operare macOS.
- **IDLE** – mediu de dezvoltare destinat dezvoltării programelor Python; autorul său este și autorul limbajului Python, deci este vorba de un mediu care este parte integrantă a distribuțiilor Python.

În imaginea 3.19. puteți vedea unul dintre mediile de dezvoltare în acțiune. Este vorba despre mediul de dezvoltare Android Studio.



Imaginea 3.19. Android Studio

Depanatoarele

Instrumentele indispensabile de dezvoltare software sunt și depanatoarele. Pe baza numelui, se poate presupune că acestea sunt instrumente care ajută la eliminarea erorilor din codul sursă al programului. Depanatoarele realizează acest lucru asigurând analiza și controlul execuției codului.

Depanatoarele există ca instrumente autonome (*standalone* în engleză), dar sunt, de asemenea, părți integrante ale oricărui mediu de dezvoltare modern.

Testarea codului de program în acest curs

În rândurile anterioare ați avut ocazia să vedeți două programe simple, unul scris în limbajul Java și celălalt în limbajul Python. Acesta arată cum este scris, compilat și rulat codul sursă. În timpul acestui curs, aveți posibilitatea de a scrie și rula codul scris folosind acele două limbaje de programare direct în browserul web, folosind compilatori integrați, adică medii de lucru care permit execuția codului. Pentru început, iată un astfel de mediu pentru execuția codului Java:

```
public class JavaHelloWorld
{

    public static void main(String[] args)
    {

        System.out.println("Hello World");

    }

}
```


Încercați să rulați programul și veți vedea că se obține același efect ca atunci când programul este rulat direct pe computer.

Programul Python din această lecție în mediul de lucru:

```
print('Hello World')
```

În lecțiile următoare, oriunde are sens, veți putea găsi aceste medii de execuție a codului pe care, folosindu-le, veți putea testa și modifica exemple.