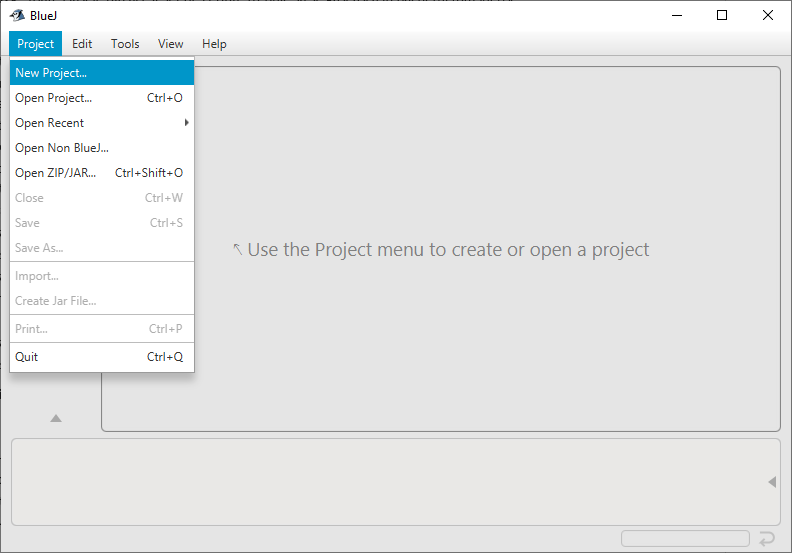
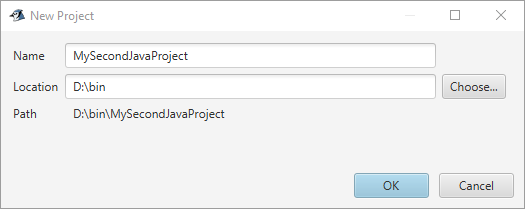
Acesta este un ghid pentru rezolvarea cerintelor Temei 1 de laborator. Pentru exemplificare am folosit programul BlueJ. Orice versiune mai mare de Java 8 este buna pentru utilizarea la aceasta tema.

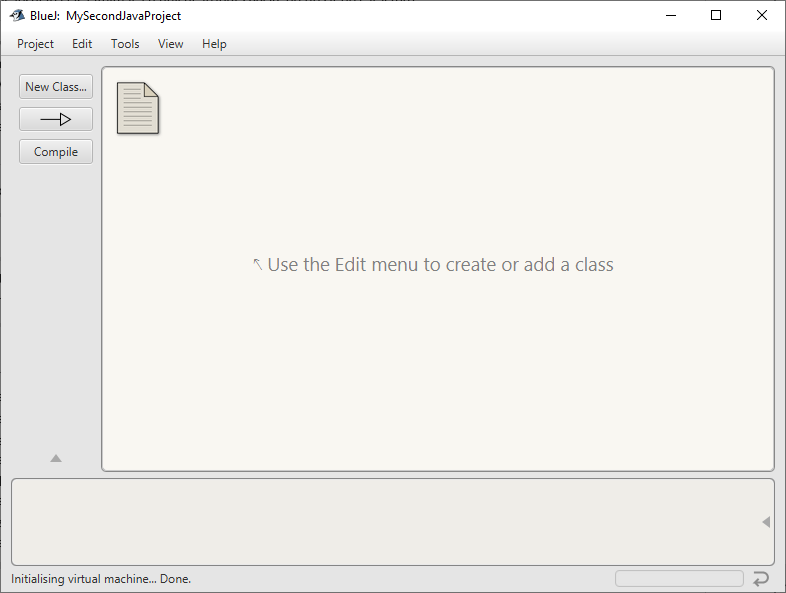
**A.** Creati proiectul MySecondJavaProject

Pentru a crea un proiect nou in BlueJ se alege din meniu *Project/New Project* si se introduce numele proiectului in pop-up-ul aparut.



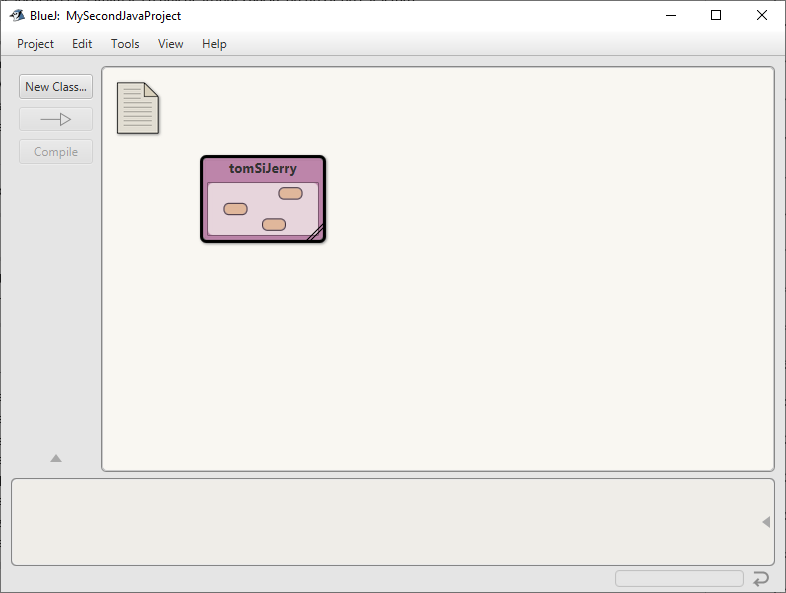


BlueJ va deschide automat noul proiect, momentan continand doar un fisier README.TXT generat automat. Se observa numele proiectului in bara de sus. In continuare ne vom referi la spatiul in care se gaseste fisierul README.TXT si in care se vor gasi clase si pachete drept “spatiu de lucru”.

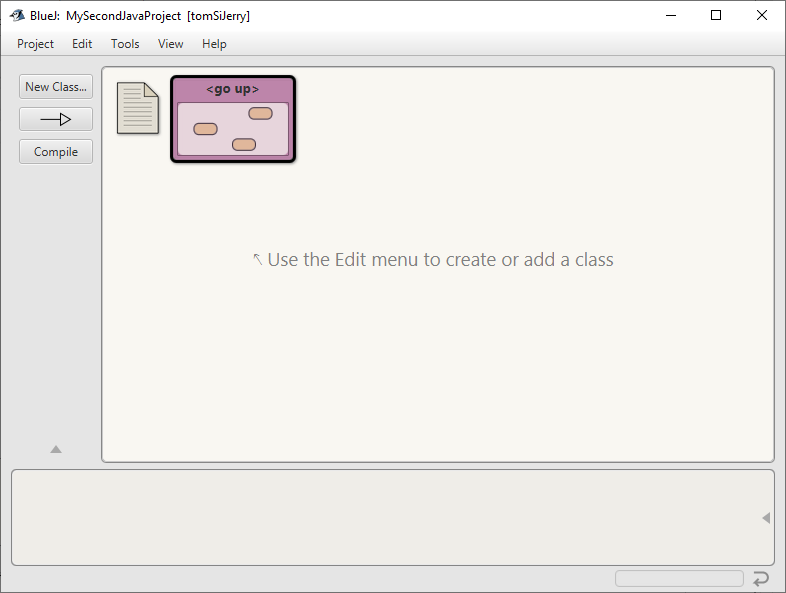


**B. Tom & Jerry** - toate clasele din acest set de cerinte trebuie sa se gaseasca in pachetul *tomSiJerry*

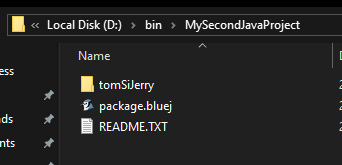
Pentru adaugarea unui nou pachet in BlueJ se da click dreapta in spatiul de lucru, se alege optiunea *New Package* si se introduce numele pachetului dorit in pop-up-ul deschis. In final se apasa OK.



Se observa reprezentarea grafica specifica a pachetului. Momentan pachetul contine doar un fisier README.TXT generat automat. Se acceseaza pachetul prin dublu click, deschizandu-se intr-o fereastra noua spatiul de lucru al pachetului. Se observa prezenta unui shortcut pentru intoarcerea la proiectul de baza. Se observa prezenta numelui pachetului in bara de sus a ferestrei.

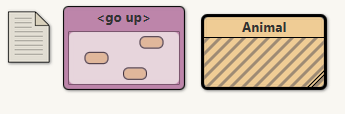


Un pachet este o colectie de clase grupate, de obicei, in functie functionalitatile sau de domeniul lor. In limbajul Java exista adesea o superpozitie intre pachete si bibliotecile software. Cand dorim sa folosim o biblioteca software sau o componenta a unei biblioteci software, vom importa un pachet sau o componenta a unui pachet. Pe disc, pachetele sunt reprezentate ca foldere.

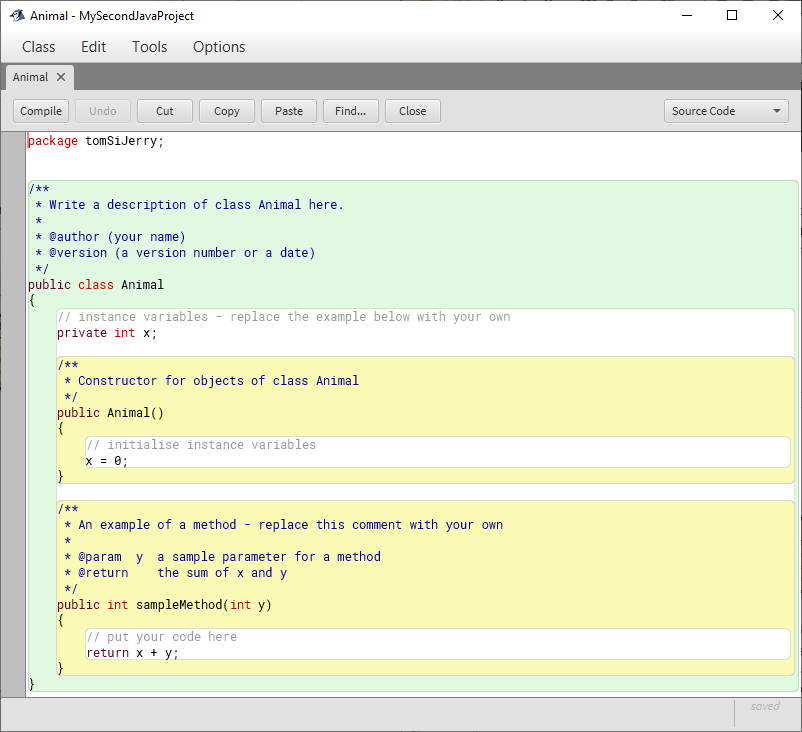


**1.** Creati si incapsulati clasa Animal, cu unicul atribut *specie*, de tip sir de caractere.

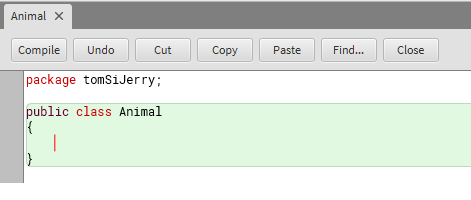
Pentru a adauga o clasa noua putem folosi butonul *New Class* din stanga spatiului de lucru sau putem da click dreapta pe spatiul de lucru si vom alege optiunea *New Class*. In pop-up-ul deschis vom introduce numele clasei, ne vom asigura ca *Class Language* are valoarea *Java* si ca *Class Type* are valoarea *Class*, dupa care vom apasa OK. Observam ca in pachet a aparut clasa Animal. Hasura gri ne anunta ca respectiva clasa nu a fost inca compilata.



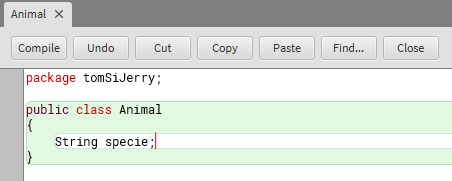
Compilarea este procedeul prin care codul scris de programator intr-un limbaj de programare este “tradus” intr-un cod pe care calculatorul il poate intelege (interpreta). Putem compila toate clasele proiectului apasand butonul *Compile* din partea stanga a spatiului de lucru. Alternativ, in editorul fiecarei clase, avem butonul *Compile* in partea de sus sau putem da click dreapta pe clasa in spatiul de lucru si alege optiunea *Compile*.



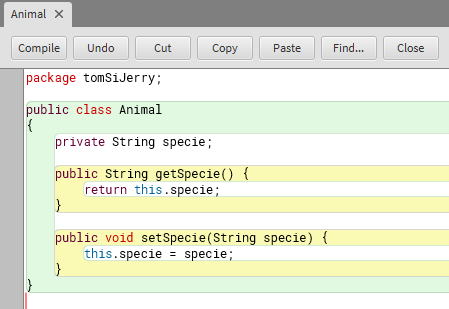
Deschizand clasa Animal pentru editare observam ca BlueJ a generat automat niste cod pentru clasa. Observam specificarea pachetului din care face parte clasa pe primul rand al fisierului. Daca ne uitam pe disc, vom vedea ca fisierul are extensia *.java*. Stergem continutul cu exceptia declararii pachetului si declararii clasei. Restul nu ne intereseaza in acest moment.



O clasa este un prototip, un model prin care putem descrie un grup de obiecte. Ganditi-va la un marker. Este un instrument de scris, cu pasta, de forma cilindrica, cu un capac din plastic. Toate aceste informatii descriu, la modul general, un marker, fiind atributele lui. Acum ganditi-va la un marker de culoare rosie, la un alt marker de culoare albastra si la un altul de culoare verde. Toate sunt markere, toate sunt instrumente de scris, cu pasta, de forma cilindrica, cu un capac din plastic. Totusi, exista deosebiri intre ele, exista ceva care le da o identitate proprie, anume culoarea. Deci, cele trei markere sunt instante ale clasei marker, fiecare cu valoare proprie a atributului culoare.

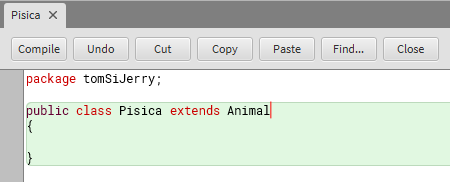


Incapsularea este un procedeu specific de ascundere a unor informatii dintr-o clasa. Presupune declararea atributelor clasei ca fiind *private* si construirea de *getters* si *setters* pentru atribute. Observati utilizarea cuvantului cheie *this* pentru a face referire la atribute ale clasei curente (this.specie se refera la acel specie declarat pe randul 5). Compilati clasa.

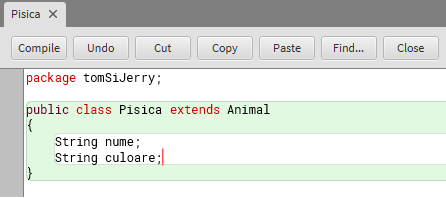


**2.** Pisica este un animal. Creati clasa Pisica cu atributele specifice *nume* si *culoare*, de tip siruri de caractere. Adaugati un constructor cu parametrii.

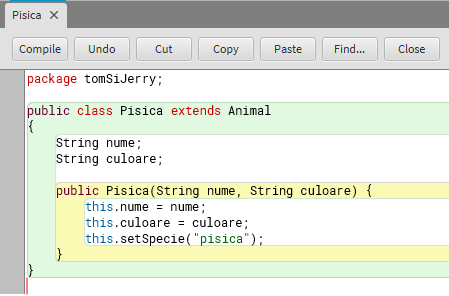
Pisica este un animal. Aici avem conceptul de mostenire. Mostenirea este procedeul prin care o clasa mosteneste atributele si comportamentele unei clase parinte (super-clasa). Mostenirea se specifica prin cuvantul cheie *extends*. In acest moment, Pisica are, la randul ei, o specie, precum si metodele getSpecie si setSpecie.



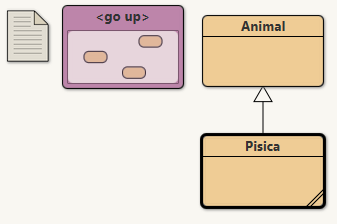
Pisica are si niste atribute specifice, care se regasesc doar in aceasta, nu si in super-clasa.



Constructorul este o metoda speciala pe care programatorul o poate folosi pentru a instantia obiecte. In mod implicit, orice clasa beneficiaza de un constructor fara parametrii. Constructorii vor avea obligatoriu numele clasei. Pentru Pisica am definit un constructor cu parametrii *nume* si *culoare*. Bineinteles, specia va fi mereu “pisica”. Observati utilizarea *setter-*ului intrucat specia a fost declarata *private* in clasa Animal.

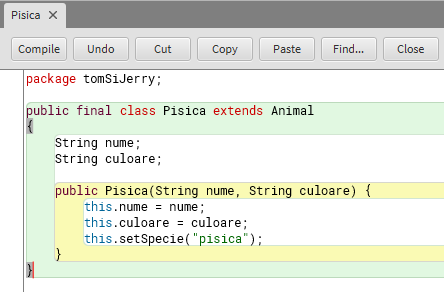


Dupa compilarea clasei Pisica putem observa in spatiul de lucru reprezentarea grafica a mostenirii prin acea sageata cu varf plin.

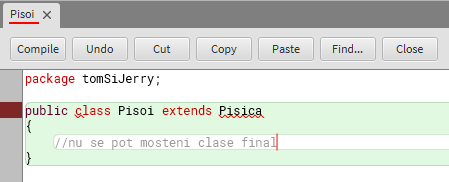


**3.** Pentru clasa Pisica adaugati modificatorul *final*. Creati clasa Pisoi si mosteniti clasa Pisica. Scrieti in clasa Pisoi un comentariu cu ce observati

Modificatorul *final* (la randul lui, cuvant cheie)se adauga inaintea cuvantului cheie *class.* Dupa compilare, observati ca nu s-a schimbat nimic in relatia Pisica-Animal.

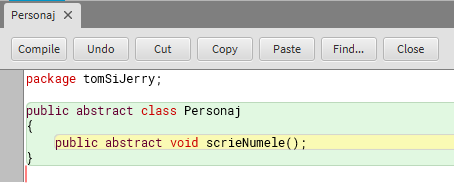


Adaugam clasa Pisoi si incercam sa mostenim clasa Pisica. Observam ca primim o eroare de compilare. Clasele marcate *final* nu pot fi mostenite. Observam hasura rosie a clasei Pisoi in spatiul de lucru. Aceasta ne arata prezenta unor erori care blocheaza compilarea respectivei clase.



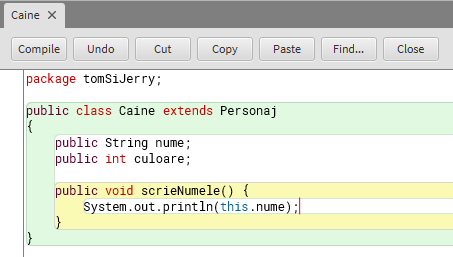
**4.** Definiti clasa abstracta Personaj cu comportamentul *scrieNumele*

Clasele abstracte sunt acele clase care contin metode abstracte. Metodele abstracte sunt sabloane de metode, stabilind doar semnaturile metodelor (tipul de data returnat, numele metodei, numarul si tipurile de parametrii). Clasele concrete care mostenesc clase abstracte trebuie sa suprascrie toate metodele abstracte. Clasele si metodele abstracte se marcheaza prin cuvantul cheie *abstract*.

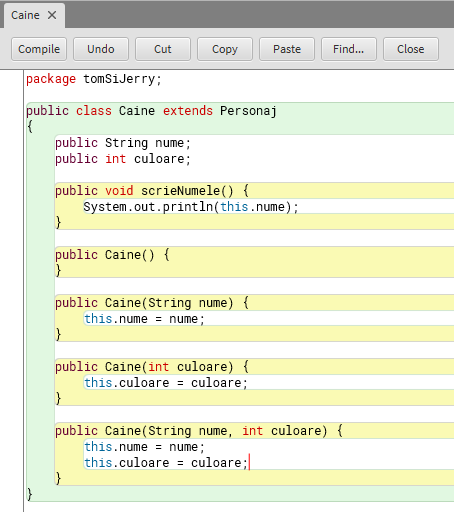


**5.** Cainele este un personaj identificat prin *nume* si *culoare.* Definiti 4 constructori ai clasei

Cainele este un personaj, deci vom defini clasa Caine care va mosteni clasa abstracta Personaj, definita anterior. Cainele este identificat prin nume si culoare, deci vom adauga atributele *nume* si *culoare* in clasa Caine. Suplimentar, mostenind clasa abstracta Personaj, suntem nevoiti sa suprascriem metoda *scrieNumele.* Suprascrierea metodei presupune copierea exacta a semnaturii metodei si crearea unui corp de instructiuni pentru metoda.

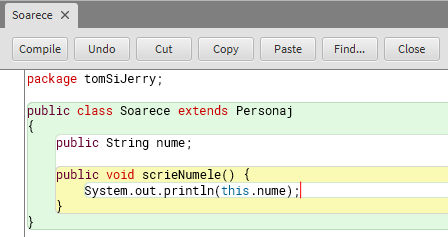


Pentru culoare am ales tipul de data *int* pentru a-mi permite definirea celor 4 constructori diferiti. Astfel, vom avea constructorul fara parametrii, constructorul cu un parametru de tip String, constructorul cu un parametru de tip int si constructorul cu 2 parametrii, String si int. Daca am fi pus tipul de date *String* pentru culoare, am fi avut eroare daca incercam sa definim constructor cu unic parametru culoare dupa ce deja am definit constructorul cu unic parametru nume. Asta deoarece compilatorul verifica semnaturile metodelor. Am fi avut doua metode (constructori) cu aceeasi semnatura *public Caine(String …)*. Numele parametrilor nu intra in semnatura, doar tipurile lor.

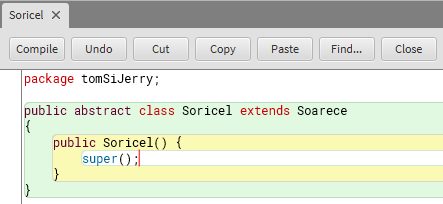


**6.** Soarecele este un personaj identificat prin *nume.* Soricel este o clasa abstracta ce extinde Soarece. Definiti in clasa Soricel un constructor in care sa apelati constructorul clasei parinte

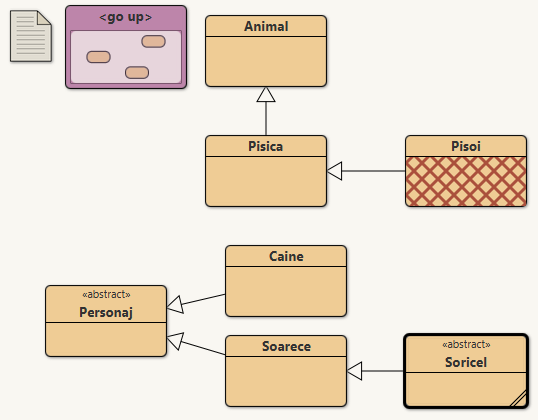
Clasa Soarece trebuie, la randul ei, sa mosteneasca Personaj. Adaugam atributul *nume* si suprascriem metoda abstracta *scrieNumele*.



Apelul constructorului clasei parinte se efectueaza cu cuvantul cheie *super*. Apelul trebuie sa fie obligatoriu pe prima linie in constructorul clasei copil. Adaugam modificatorul *abstract* clasei Soricel.

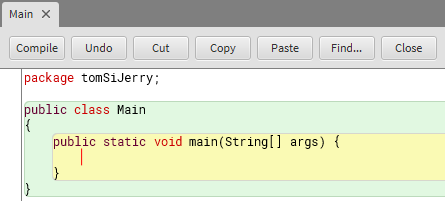


Asa arata spatiul nostru de lucru acum.



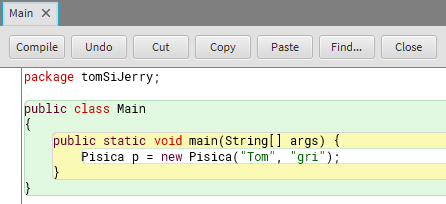
**7.** Adaugati clasa Main. In metoda *main:*

Adaugam clasa Main si adaugam in ea metoda *main* sub forma cunoscuta.



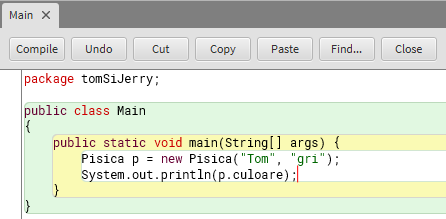
- instantiati o pisica cu numele Tom si culoarea gri

Pentru instantiere se foloseste cuvantul cheie *new*.



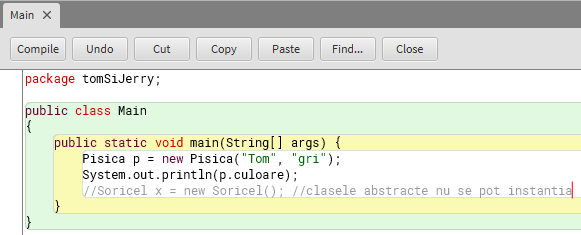
- afisati in consola culoarea pisicii Tom

Pentru a afisa culoarea folosim *System.out.println*. System ne da acces la fluxurile de intrare-iesire ale sistemului, out reprezinta fluxul de iesire, iar println este metoda prin care se scrie in fluxul de iesire (terminal/consola). Pentru a accesa culoarea pisicii *p* vom folosi operatorul de dereferentiere .



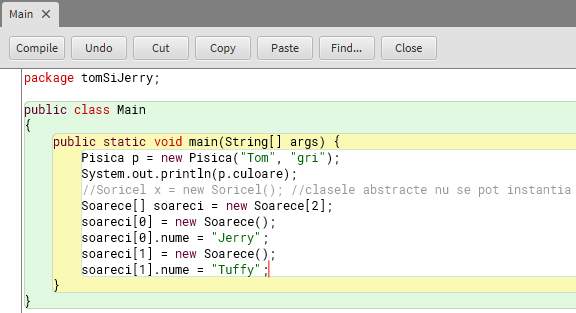
- instantiati un Soricel si adaugati un comentariu cu ce observati

Clasele abstracte nu se pot instantia.



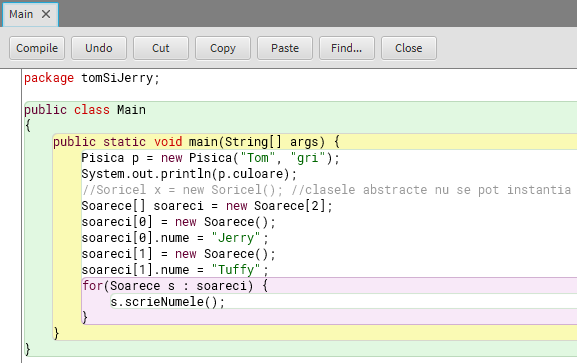
- definiti un vector cu elemente Soarece in care adaugati soarecii Jerry si Tuffy

Prin vector se intelege un array []. Array-ul are intotdeauna o dimensiune fixa, in cazul nostru 2. Indecsii incep intotdeauna de la 0;



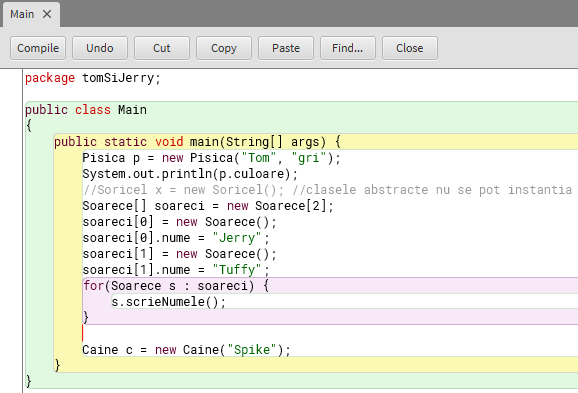
- utilizati o bucla for pentru a afisa in consola numele soarecilor

Fiind obiecte, putem folosi sintaxa for-each. Altfel, putem parcurge cu un index *int i*.



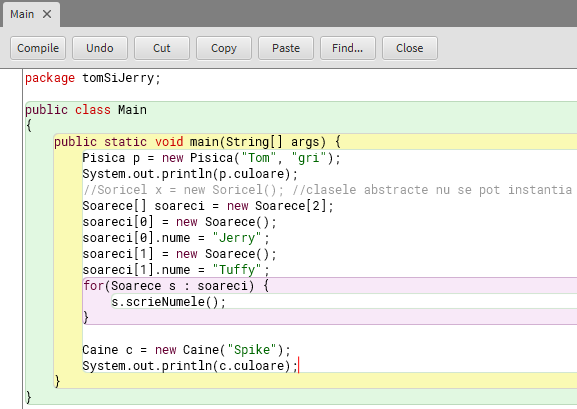
- instantiati cainele Spike folosind constructorul cu unic parametru *nume*

Aici era un indiciu pentru punctul 5, anume ca trebuie definit un constructor cu unic parametru *nume* in clasa Caine.

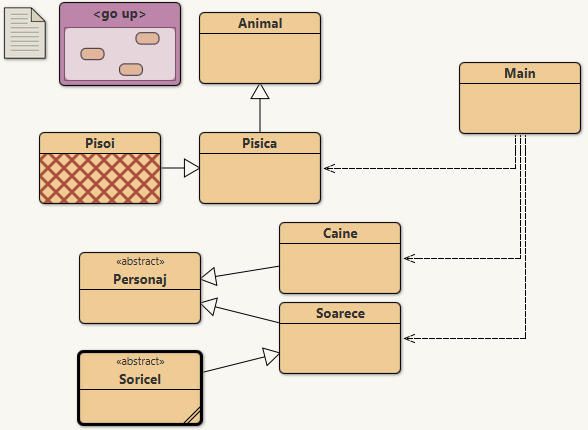


**-** afisati culoarea lui Spike

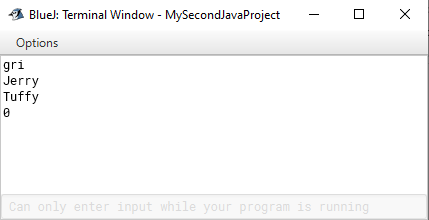
Putem folosi din nou operatorul de dereferentiere . pentru a afisa culoarea cainelui.



Observati forma finala a spatiului de lucru. Instantierile sunt marcate in diagrama prin sageti cu linie intrerupta.



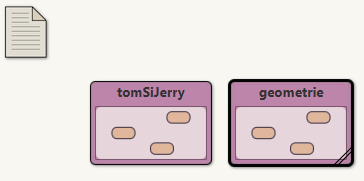
Pentru a rula metoda *main* putem da click dreapta pe clasa Main, alegem metoda (cel mai probabil al doilea rand din meniul deschis) si apasam OK.



Culoarea lui Spike este 0 fiind valoarea default a unui atribut *int*. In clasa Caine am definit culoarea ca fiind int, iar in constructorul apelat pentru a-l instantia pe Spike nu am specificat nici o culoare. Daca am fi folosit pentru culoare un obiect (de exemplu, String), culoarea lui Spike ar fi fost *null* (cuvant cheie ce semnifica absenta unei instante).

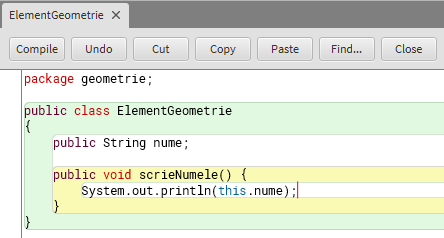
**C. Geometrie** - toate clasele din acest set de cerinte trebuie sa se gaseasca in pachetul *geometrie*

Ne intoarcem in spatiul de lucru al proiectului (dublu click pe ‘go up’ in pachetul *tomSiJerry*) si adaugam un nou pachet *geometrie*.



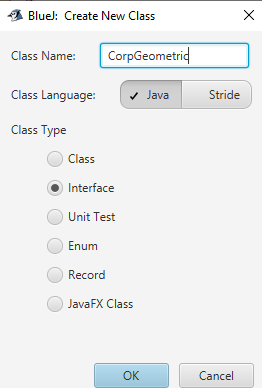
**1.** Creati clasa ElementGeometrie cu atrbutul *nume* si metoda *scrieNume.*

Intram in spatiul de lucru al pachetului geometrie si cream clasa ElementGeometrie.

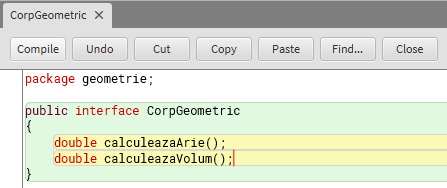


**2.** Creati interfata CorpGeometric cu metodele *calculeazaArie* si *calculeazaVolum*

Pentru a crea o interfata, in pop-up-ul de adaugare clasa putem alege *Class Type* = *Interface*.

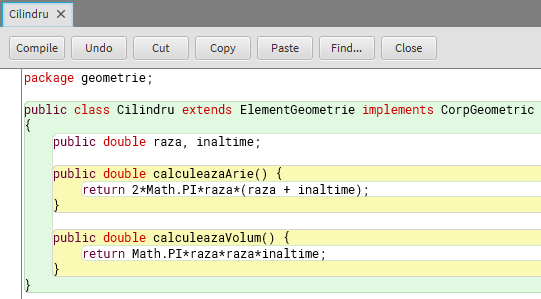


Interfata este un tip special de clasa abstracta in care toate metodele sunt implicit publice si abstracte. Este semnalizata prin cuvantul cheie *interface*.

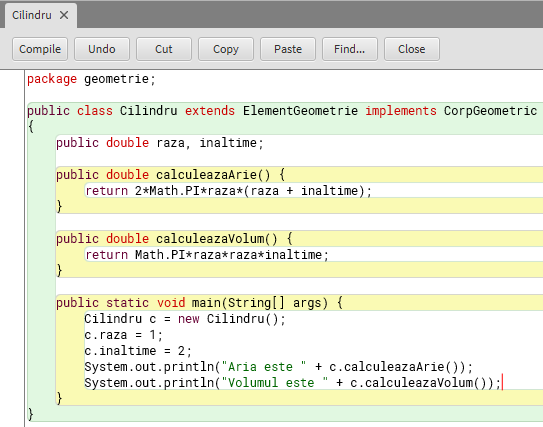


**3.** Cilindrul este atat element de geometrie, cat si un corp geometric. Adaugati in clasa cilindru metoda *main*. Instantiati un cilindru cu raza 1 si inaltimea 2 si afisati-i aria si volumul.

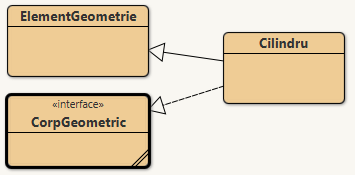
In Java nu exista mostenire multipla, insa putem mosteni o clasa si implementa una sau mai multe interfete simultan. Specificam implementarea unei interfete folosind cuvantul cheie *implements*. Fiind abstracte, trebuie sa suprascriem (implementam) toate metodele interfetei. Folosim cuvantul cheie *return* pentru a spune metodei sa “emita” o valoare conform tipului de data din semnatura ei.



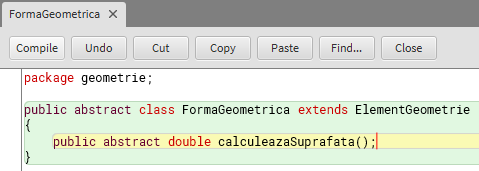
Putem folosi constanta PI din utilitarul Math sau putem defini propria constanta. Cum probabil banuiati deja, metoda *main* poate fi definita in orice clasa, nu neaparat intr-o clasa numita Main.



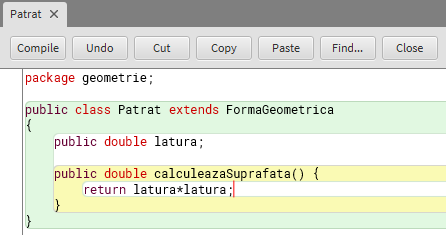
In spatiul de lucru putem observa reprezentarea implementarii interfetei printr-o sageata intrerupta cu cap plin.



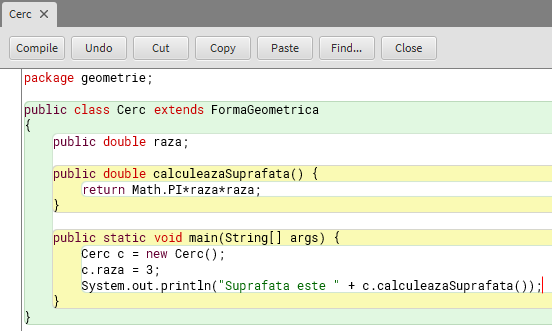
**4.** Definiti clasa abstracta FormaGeometrica stiind ca formele geometrice sunt elemente de geometrie pentru care putem calcula suprafata.



**5.** Creati clasele Patrat si Cerc stiind ca acestea sunt forme geometrice. In clasa Cerc adaugati metoda *main* in care instantiati un cerc cu raza 3 si afisati aria.

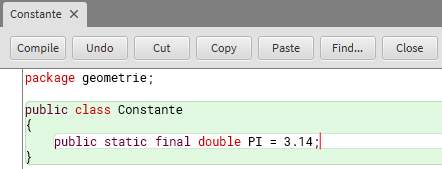


Proiectul ne permite chiar sa definim mai multe metode *main*. In acest caz, la pornirea proiectului, masina virtuala trebuie “anuntata” din care metoda *main* sa inceapa executia.

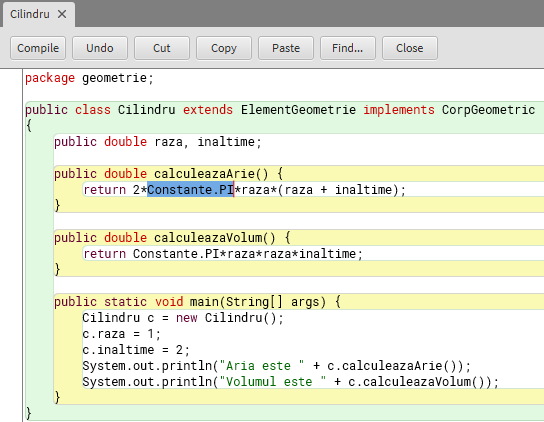


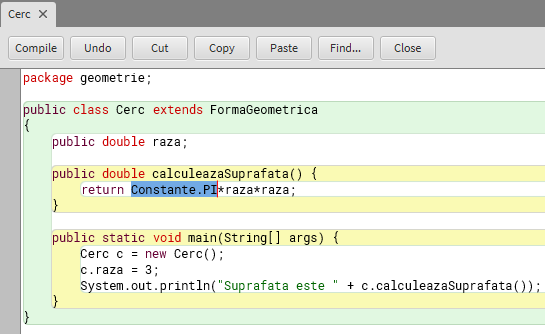
**6.** Mutati constanta *PI* intr-o clasa noua Constante si folositi-o de acolo.

Adaugam o clasa Constante in care adaugam o constanta numita PI, folosind cuvantul cheie *final*. Constanta PI trebuie sa fie statica pentru a o putea folosi fara a instantia un obiect din clasa Constante.



Ne intoarcem in clasele cilindru si cerc si folosim constanta nou definita (in loc de Math.PI).





In final, spatiul de lucru al pachetului *geometrie* arata astfel.

