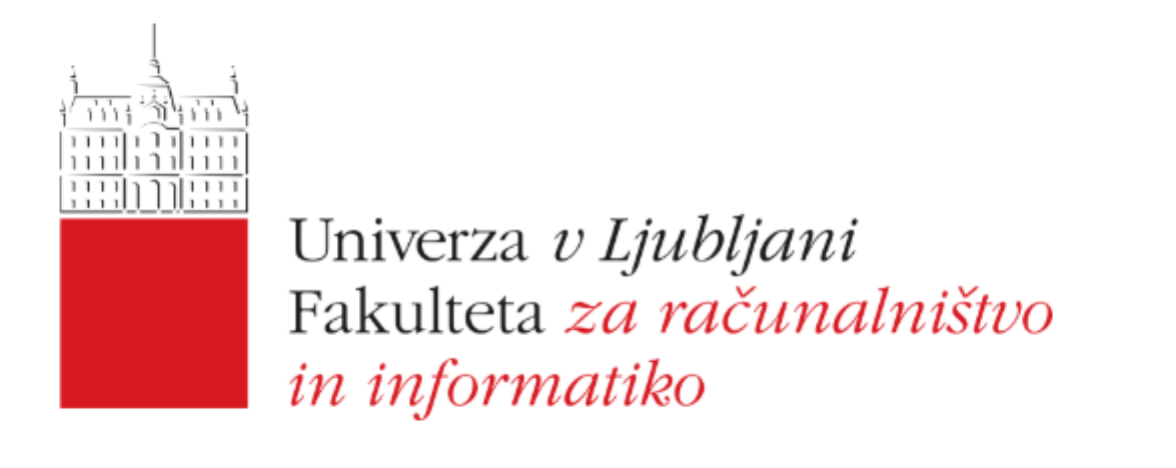


Opis sistema in osnovna navodila za uporab

Tomaž Dobravec



KAZALO

1 Opis sistema 3

1.1 Namen 3

2 Delovanje sistema 3

2.1 ALGator kot samostojna aplikacija 3

2.2 ALGator kot spletna aplikacija 4

2.2.1 Vloge pri uporabi spletne aplikacije ALGator 4

2.2.2 Primer uporabe spletne aplikacije ALGator 5

3 Izvajanje algoritmov 6

3.1 Definicija osnovnih pojmov 6

3.2 Naloge in množice nalog 7

3.3 Algoritmi 10

3.4 Rešitev naloge in zapis rezultatov 13

3.4.1 Privzeti izhodni parametri in indikatorji 14

3.5 Meritve 14

3.5.1 Meritve časovnih in specifičnih indikatorjev izvajanja 14

3.5.2 Meritve s števci 15

3.5.3 Meritve s prirejenim JVM 16

3.6 Izdelava projekta 17

4 Konfiguracija sistema 19

# Opis sistema

ALGator je sistem za izvajanje algoritmov na podanih testnih podatkih ter analizo rezultatov izvajanja.

Sistem omogoča dodajanje in upravljanje s poljubnim številom projektov. V okviru enega projekta je definiran problem, testne množice vhodnih podatkov ter način reševanja nalog tega problema. Projekt lahko vsebuje poljubno število algoritmov, ki naloge rešujejo na predpisan način. Sistem omogoča analizo izvajanja posameznega algoritma ter primerjavo med algoritmi istega projekta.

## Namen

* Reševanje definiranih problemov z različnimi algoritmi.
* Analiza delovanja posameznega algoritma pri reševanju nalog danega problema.
* Primerjava različnih algoritmov za reševanje istega problema.

# Delovanje sistema

ALGator lahko uporabljamo na dva načina: kot samostojno ali kot spletno aplikacijo. Samostojna aplikacija je namenjena razvijalcem, ki želijo ALGator uporabljati na svojem računalniku, ločeno od ostalih uporabnikov. Spletna aplikacija omogoča uporabo sistema več uporabnikom hkrati. Rezultati izvajanja spletne aplikacije so vidni vsem uporabnikom.

## ALGator kot samostojna aplikacija

ALGator je zbirka javanskih razredov, ki so združeni v knjižnici ALGator.jar. Za pravilno delovanje sistema, je potrebno:

1. ustvariti lokalni mapi algator\_root in data\_root (priporočeno je, da se mapa data\_root nahaja znotraj mape algator\_root);
2. pridobiti program ALGator (ALGator.jar + folder lib) in ga posneti v mapo app (ki naj se nahaja znotraj mape algator\_root);
3. pravilno nastaviti sistemske spremenljivke
   * + CLASSPATH (dodati je treba pot do ALGator.jar) in
     + ALGATOR\_ROOT (kazati mora na mapo algator\_root);
4. ustvariti vsaj en projekt (glej poglavje 3.6).

Zadnja verzija programa ALGator se nahaja na https://github.com/ALGatorDevel/Algator. Tam je v datoteki README opisan postopek namestitve.

ALGator izvajamo z enim od naslednjih ukazov:

**java algator.Version**

izpis sporočila o verziji ALGatorja in privzeti nastavitvi folderjev

ALGATOR\_ROOT, ALGATOR\_DATA\_ROOT in ALGATOR\_DATA\_LOCAL

**java algator.Admin**

ustvarjanje konfiguracijskih datotek novega projekta/algoritma

**java algator.Execute**

izvajanje definiranih algoritmov izbranega problema

**java algator.Analyse**

analiza rezultatov izvajanja

Uporabnik ALGatorja kot samostojne aplikacije ima popoln nadzor nad celotno konfiguracijo sistema, saj hkrati nastopa v vlogi skrbnika sistema, skrbnika projekta in raziskovalca (za pomen posamezne vloge glej opis spodaj).

## ALGator kot spletna aplikacija

Za pravilno delovanje ALGator spletne aplikacije potrebujemo tri podsisteme: **AEE** (ALGator execution engine), **ADE** (ALGator data engine) in **AWE** (ALGator Web engine). Na posameznem fizičnem računalniku lahko teče več podsistemov (na primer: en fizični računalnik je lahko AEE ADE in AWE hkrati). Spletna aplikacija vsebuje natanko en ADE in en AWE ter enega ali več AEE.

### Vloge pri uporabi spletne aplikacije ALGator

Spletni sistem ALGator lahko uporabljajo uporabniki z različnimi vlogami in pravicami. Neprijavljen uporabnik (gost) lahko izvaja le funkcije, ki ne spreminjajo podatkov sistema (podatke lahko le pregleduje). Za izvajanje ostalih funkcij mora biti uporabnik v sistem prijavljen. Natančni pomen posamezne vloge je opisan v nadaljevanju.

#### Skrbnik sistema

Skrbnik sistema postavi celotni sistem in skrbi za strojno in programsko opremo in ima vse pravice in dostop do vseh virov sistema.

#### Skrbnik projekta

Skrbnik projekta definira en projekt. K definiciji projekta sodi:

* definicija problema,
* definicija testnih množic vhodnih podatkov,
* opis formata vhodnih in izhodnih podatkov,
* definicija vseh javanskih razredov, ki so potrebni za izvajanje algoritmov tega projekta.

Skrbnik projekta ima dostop do vseh virov projekta. Če je projekt javen (ta atribut nastavi skrbnik projekta), lahko podatke o projektu vidijo vsi uporabniki. Privatne projekte vidi samo skrbnik sistema in skrbnik projekta.

#### Raziskovalec

Raziskovalec definira en algoritem znotraj izbranega projekta. Javne algoritme (ta atribut nastavi raziskovalec) lahko vidijo vsi uporabniki, privatnega pa poleg skrbnikov le raziskovalec.

#### Gost

Gost lahko vidi podatke o javnih algoritmih javnih projektov.

Za podrobnejše informacije o delovanju spletne aplikacije glej ALGator-dev.docx

### Primer uporabe spletne aplikacije ALGator

V tem razdelku je opisan najpogostejši način uporabe spletne aplikacije ALGator.

* Skrbnik sistema postavi celotni sistem in objavi povezavo do spletne strani sistema.
* Skrbnik projekta doda nov projekt; s tem definira problem in nabore testnih primerov ter predpiše način reševanja tega problema. Za vsak definiran projekt sistem avtomatsko ustvari spletno stran z razlago problema in navodili za pripravo algoritma.
* Skrbnik projekta v projekt doda nekaj znanih algoritmov za reševanje tega problema, ki bodo služili za osnovno primerjavo z novimi algoritmi, ki jih bodo dodajali raziskovalci.
* Raziskovalec skladno z navodili, objavljenimi na spletni strani projekta, pripravi nov algoritem in ga doda v sistem. ALGator dodani algoritem avtomatsko požene na testnih podatkih. Raziskovalec nato primerja učinkovitost tega algoritma z učinkovitostjo algoritmov, ki so že v projektu. Če želi, lahko algoritem odpre za javnost (algoritem postane javen) ali pa ga skrije (privaten algoritem). Privatne algoritme lahko raziskovalec kadarkoli spremeni v javne.
* Gost sistema lahko pregleduje podatke o vseh javnih projektih in javnih algoritmih. Lahko izvaja primerjave in ostale operacije, ki ne spreminjajo konfiguracije sistema. Če želi, se lahko kadarkoli prijavi kot raziskovalec ali kot skrbnik projekta.

# Izvajanje algoritmov

## Definicija osnovnih pojmov

* **KONFIGURACIJSKA DATOTEKA**

Vse entitete sistema ALGator so opisane v konfiguracijskih datotekah (ena datoteka opisuje eno entiteto). Format konfiguracijskih datotek je JSON.

Primer: izsek datoteke, ki opisuje projekt:

{

"Project" : {

"Name" : "Sorting",

"Description" : "Internal sorting of integer arrays",

"Algorithms" : ["Hoare","Wirth"],

"TestSets" : ["TestSet1","TestSet2"],

...

}

}

* **SPREMENLJIVKA (variable)**

Spremenjlivka, ki jo uporabljamo za opis neke lastnosti. Vsaka spremenljivka ima svoje ime, tip in vrednost. Spremenljivka je lahko zapisana v konfiguracijski datoteki ali pa »živi« znotraj javanskega razreda.

* **PARAMETER**

Spremenljivka, ki opisuje lastnost naloge. Primer: parameter N opisuje velikost tabele, ki jo želimo urediti.

* **INDIKATOR**

Spremenljivka, ki opisuje lastnost izvajanja algoritma pri dani nalogi. Primer: indikator Tmin opisuje čas izvajanja algoritma pri urejanju tabele.

* **PROBLEM**

Problem, ki ga rešujemo. Primer: urejanje podatkov, trgovski potnik, stiskanje datotek, linearno programiranje, …

* **PROJEKT**

Projekt je skupek konfiguracijskih datotek, s katerimi v sistemu ALGator opišemo problem in pripadajoče algoritme.

* **NALOGA (test case)**

Naloga danega problema. Primer za problem urejanja podatkov: ena tabela, ki jo je potrebno urediti.

* **MNOŽICA NALOG (test set)**

Zbirka naloge, ki predstavljajo najmanjšo enoto reševanja (ob enem zagonu sistema ALGator rešimo eno množico nalog).

* **ALGORITEM**

Javanski razred, ki reši eno nalogo danega problema.

* **IZVRŠEVALEC (executor)**

Orodje, s katerim poženemo dani algoritem nad dano množico nalog. Gre za javanski razred, ki je zapakiran v JAR datoteko. Izvrševalec poženemo iz ukazne vrstice; pri tem podamo parametre, ki opisujejo projekt, algoritem in množico nalog. Primer klica:

java algator.Execute Sorting -a QuickSort -t TestSet1

(ob izvršitvi tega klica se bo ALGator ”sprehodil” čez vse naloge testne množice TestSet1 in z vsako nalogo pognal algoritem QuickSort)

* **REŠITEV NALOGE**

Skupek parametrov in indikatorjev, ki opisujejo obnašanje algoritma pri reševanju dane naloge (velikost problema, poraba časa, pravilnost rezulata, poraba virov, ...)

* **REŠITEV MNOŽICE NALOG**

Datoteka, ki vsebuje rešitve za vse naloge dane množice nalog v CSV obliki: rešitev vsake naloge je zapisana v svoji vrstici, parametri in indikatorji znotraj vrstice so ločeni s podpičjem. Vrstni red parametrov in indikatorjev je določen z ustrezno atrd datoteko.

## Naloge in množice nalog

* ALGator z danim algoritmom reši vse naloge dane množice nalog.
* Za opis ene naloge v ALGatorju uporabljamo razred AbstractInput, ki vsebuje osnovne podatke o vhodu (ime naloge, velikost naloge, ...). Ker se opisi nalog posameznih problemov med seboj močno razlikujejo, mora skrbnik projekta napisati razred

class [Project]Input extends AbstractInput {…}

v katerem predvidi podatkovne strukture za shranjevanje podatkov o nalogi.

Primer: pri projektu Sorting (urejanje podatkov) za shranjevanje podatkov v razredu SortingInput definiram tabelo

public int [] arrayToSort;

* Podobno kot za vhod tudi za izhod algoritma velja, da je močno odvisen od problema. Zato mora skrbnik projekta napisati tudi razred

class [Project]Output extends AbstractOutput {…}

v katerem definira podatkovne strukture za shranjevanje podatkov o izhodu.

V primeru projekta Sorting je izhod podoben vhodu (ena tabela), zato sta tudi razreda SortingInput in SortingOutput podobna:

class SortingOutput extends AbstractOutput {

public int [] sortedArray;

...

}

* ALGator z algoritmi, podanimi v projektu, rešuje naloge, ki so zapisane v t.i. testnih množicah. Vsaka testna množica vsebuje eno ali več nalog (število nalog v testni množici določi skrbnik projekta, običajne testne množice vsebujejo od 10 do 100 nalog). Kakšen vhod bo dobil algoritem, je odvisno od problema (primer: naloga faktorizacije je podana z eno celoštevilsko vrednostjo, naloga urejanja s tabelo števil, naloga ZIP pa je podana z datoteko), zato je tudi zapis vhoda v testnih množicah odvisen od problema. Skrbnik projekta (ki edini ve, kakšen vhod bodo dobili algoritmi tega problema) določi, kako se bo vhod posamezne naloge zapisal v testne množice.

ALGator predpostavlja naslednje:

* + Za vsako testno množico [Testset] sta v direktoriju tests danega projekta dve datoteki: [Testset].atts in [Testset].txt.
  + V direktoriju tests (ali v poddirektorijih, ki jih po lastni presoji ustvari skrbnik projekta) se nahajajo vse datoteke, ki jih potrebujemo za izvajanje testov (vhodni podatki). Ko se v konfiguracijskih datotekah sklicujemo na te datoteke, uporabimo relativne poti (glede na direktorij tests).
  + Datoteka [Testset].atts je konfiguracijska datoteka, v kateri je podano ime in opis ter število nalog te testne množice.
  + Datoteka [Testset].txt opisuje posamezne naloge - **vsaka naloga je opisana v eni vrstici te datoteke**. Ker je opis naloge močno odvisen od problema, je format te datoteke povsem odvisen od projekta. Format določi skrbnik projekta.

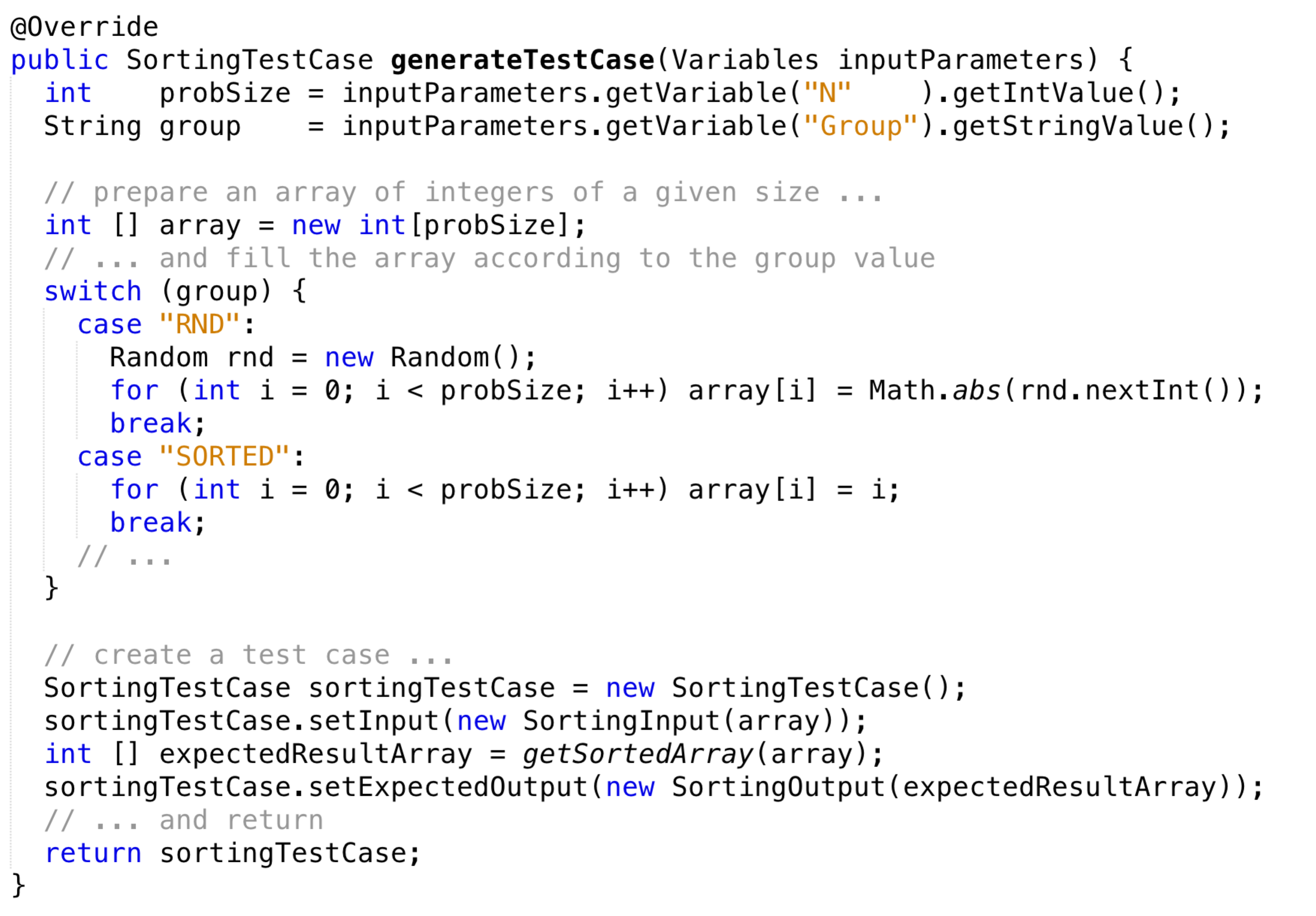
Skrbnik projekta mora predvideti, kako bo na podlagi podatkov, zapisanih v datotekah [TestSet].atts in [TestSet].txt pridobil podatke o vseh nalogah te testne množice. V ta namen mora napisati dve metodi razreda [Project]TestCase: metoda getTestCase()poskrbi, da se iz vrstice v [TestSet].txt datoteki pravilno preberejo vsi parametri, ki opisujejo test in kliče metodo generateTestCase(), ki na podlagi podanih parametrov dejansko ustvari primerek testne naloge.

Metoda [Project]TestCase.getTestCase() se kliče posredno iz metode getCurrent() razreda DefaulttestSetIterator, ki poskrbi za prehod čez celotno množico testnih primerov.



Slika 1: Primer zapisa nalog in implementacije metode getTestCase() za problem Sorting.

Primer zapisa testne množica in implementacije metode getTestCase() za projekt Sorting je prikazan na sliki Slika 1, Slika 2 pa prikazuje primer implementacije metode generateTestCase().



Slika 2: Implementacija metode generateTestCase() za problem Sorting.

## Algoritmi

* Za izvajanje algoritmov ALGator uporablja razred AbstractAlgorithm, ki vsebuje naslednje metode:
  + ErrorStatus init(AbstractTestCase test)

Metoda poskrbi, da se iz podatkov, ki so zapisani v objektu test, ustvarijo podatki, na katerih se bo izvedel test. Pri tem se morajo opraviti vsi časovno zahtevni postopki v zvezi z obdelavo vhodnih podatkov. Po koncu te metode mora biti vse pripravljeno za dejansko izvajanje algoritma.

* + void run()

V tej metodi se kliče metoda execute(<parameters>). Parametri metode execute() so odvisni od problema in jih v času ustvarjanja projekta predvidi skrbnik. **Čas izvajanja metode run() se šteje kot čas izvajanja algoritma**, zato morajo biti parametri, ki se podajo v metodi execute(), že prej skrbno pripravljeni (v metodi init()). Priprava oziroma obdelava parametrov znotraj metode run() po krivici veča časovno zahtevnost algoritma!

* + Variables done();

Metoda zbere vse parametre (vhodni parametri, parametri izvajanja in časovni parametri) in jih vrne v tabeli tipa Variables.

Razred AbstractAlgorithm je abstrakten. Naloga skrbnika projekta je, da napiše naslednika tega razred -[Project]AbsAlgorithm - in napiše metodo run() (in po potrebi razširi tudi ostale metode). V metodi run() se kliče metoda execute(), podpis za to metodo pa mora definirati skrbnik. Razen v redkih primerih (recimo takrat, ko ima algoritem več faz, ki bi jih radi obravnavali ločeno) je podpis metode execute() tak:

**protected** **abstract** [Project]Output execute([Project]Input input);

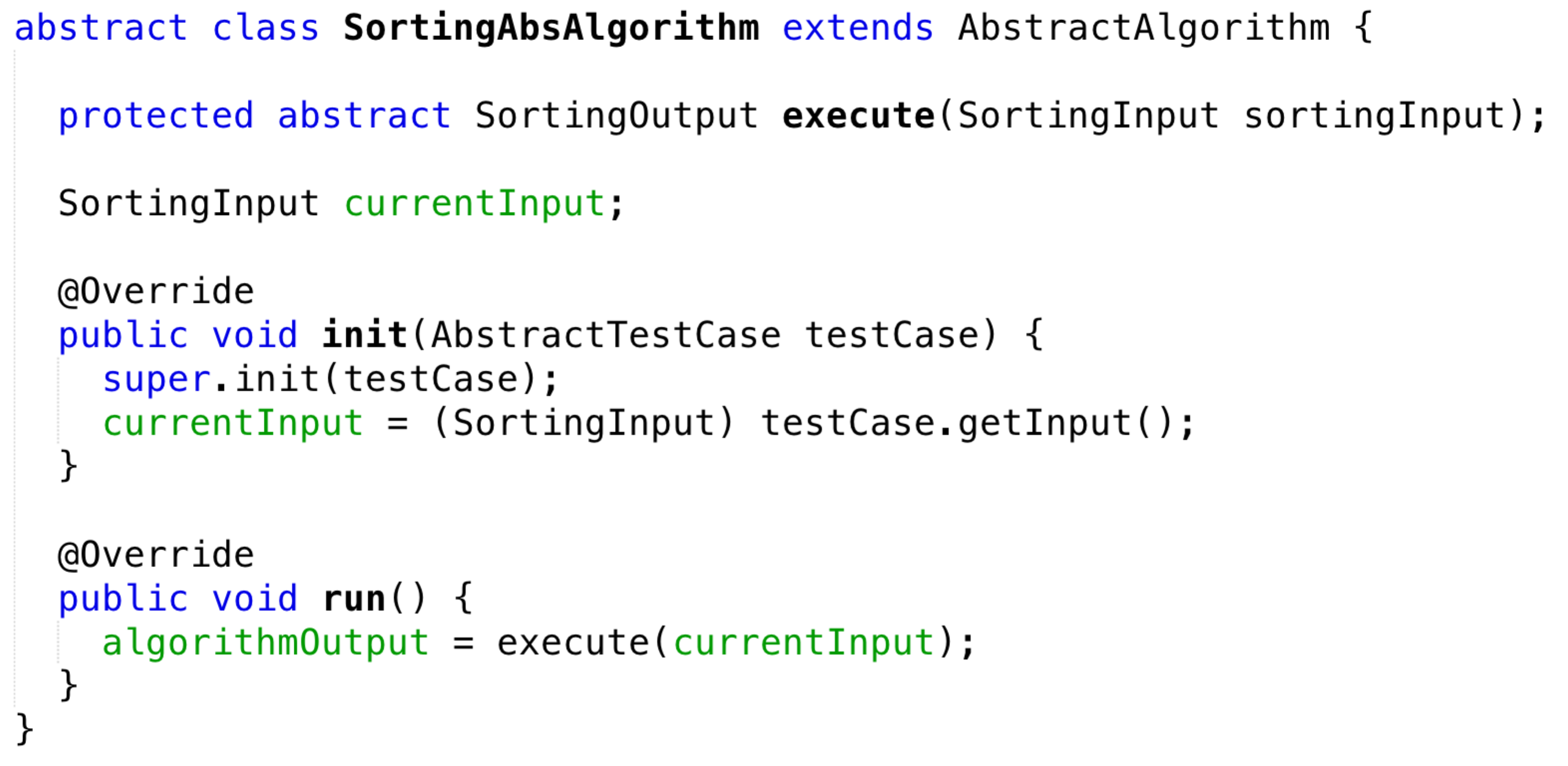
metoda run() pa vsebuje le klic, takole:

**public** **void** run() {

algorithmOutput = execute(currentInput);

}

Spodaj je prikazan primer implementacije razreda SortingAbsAlgorithm, v katerem v metodi init() poskrbimo za ”ekstrakcijo” podatkov, v metodi run() pa kličemo abstraktno metodo execute().



Naloga raziskovalca je, da implementira naslednika razreda [Project]AbsAlgorithm. Da bi preprečili morebitno zmedo, se raziskovalcu posreduje poenostavljeno verzijo tega razreda – razred, ki je direkten naslednik razreda Object in vsebuje le abstraktno metodo execute(...). Razred, ki je naslednik tega poenostavljenega razreda, bo hkrati (brez sprememb kode) tudi naslednik pravega razreda [Project]AbsAlgorithm. Na ta način raziskovalcu ni treba vedeti za zapleteno strukturo pravega očeta in se lahko osredotoči samo na implementacijo algoritma, hkrati pa bo njegov razred brez sprememb uporaben v širšem kontekstu celotnega sistema.

Primer implementacija razreda AbstractAlgorithm in njegovih naslednikov za konkreten problem je prikazan na sliki Slika 3.



Slika 3. Razred AbsAlgorithm in njegovi nasledniki

## Rešitev naloge in zapis rezultatov

Rešitev ene naloge je sestavljena iz parametrov in indikatorjev.

Parametri opisujejo lastnosti naloge (na primer: velikost problema N, skupina problema, ...). Zapisani so v datoteki, ki opisuje test ([Testset].txt). Za lažjo analizo rezultatov lahko vrednost parametrov prenesemo tudi v datoteko z rezultati testiranje (primer: parameter o velikost vhoda uporabimo za urejanje rezultatov glede na velikost vhoda ali za združevanje rezultatov z isto velikostjo vhoda).

Indikatorji opisujejo lastnosti izvajanja algoritma na danem testnem primeru. Indikatorje delimo na

* Specifični indikatorji

Specifični indikatorji opisujejo specifične lastnosti algoritma. Primer: numerična rešitev naloge, kakovost rešitve, pravilnost rešitve, ... (compresionRate, myResult/optResult, OK/NOK, ..). Vrednost specifičnih indikatorjev se določi v metodi [Project]AbsAlgorithm.done().

* Časovni indikatorji

Časovni indikatorji opisujejo čas(e) izvajanja. Osnovni časovni indikator meri čas izvajanj celotnega algoritma (čas izvajanja metode [Project]AbsAlgorithm.run()).

* Števci

Števci povejo, koilkokrat se je med izvajanjem algoritma na danem testu izvedla vrstica, ki v kodi sledi ukazu //@COUNT{counter\_name, value}. Primer: če v algoritem za urejanje podatkov pred vsako vrstico, v kateri primerjamo dva podatka, zapišemo //@COUNT{compare, 1}, bo na koncu izvajanja algoritma števec compare vseboval število vseh primerjav, ki so se izvedle.

* JVM indikatorji

Število posameznih java bytecode ukazov, ki so se izvedli med izvajanjem algoritma.

Algoritmi se običajno izvajajo nad celotno množico nalog (ob enem izvajanju se izvedejo vsi testi podane množice). Rezultati izvajanja se zapišejo v izhodno datoteko, v kateri vsakemu testu pripada ena vrstica s parametri in indikatorji. Kateri parametri se nahajajo v tej vrstici in kakšen je njihov vrstni red, je določeno s pripadajočo atrd konfiguracijsko datoteko:

* za parametre ter specifične in časovne indikatorje – [Project]-em.atrd,
* za števce – [Project]-cnt.atrd,
* za JVM indikatorje – [Project]-jvm.atrd,

### Privzeti izhodni parametri in indikatorji

Vsaka vrstica z rešitvijo vsebuje tri privzete izhodne parametre ter en indikator:

* ime algoritma,
* ime testne množice,
* ime testa,
* indikator uspešnosti testa.

Ime testa mora biti enolično za dano testno množico (dva testa znotraj iste testne množice ne smeta imeti istega imena). Če združimo več datotek z rezultati, prvi trije parametri enolično določajo test.

Indikator uspešnosti testa pove, ali se je izvajanje algoritma končalo v predvidenem času (parameter TimeLimit v atts datoteki). Indikator uspešnosti ima lahko tri vrednosti: PASS (izvajanje se je končalo brez napak), FAILED (med izvajanjem je prišlo do izjeme) ali KILLED (izvajanje je bilo ustavljeno po TimeLimit sekund).

## Meritve

Izračun indikatorjev ALGator opravi z več ločenimi meritvami (primer: z eno meritvijo izmerimo časovne, z drugo pa jvm parametre). Ena meritev se izvede nad celotno množico nalog. Rezultati vsake meritve so zapisani v samostojni datoteki. Za par (algoritem, testna množica) imamo zato več izhodnih datotek, za vsako meritev po eno.

Vsebina izhodnih datotek (katere parametre in indikatorje vsebuje) je opisana v pripadajoči atrd datoteki (proj/[Project]-[ext].atrd). Ime izhodne datoteke je [Algoritem]-[TestSetName].[ext]. Pri tem je [ext] končnica, specifična za to meritev (em, cnt in jvm).

### Meritve časovnih in specifičnih indikatorjev izvajanja

Ime atrd datoteke: proj/Project-em.atrd

Končnica: izhodne datoteke: .em

Glavni namen te meritve je izračun specifičnih ter časovnih indikatorjev.

Časovni indikatorji se izmerijo avtomatsko (čas izvajanja metode run()). Vsak test se požene večkrat (parameter TestRepeat v atts datoteki). V izhodno datoteko se zapiše ena številska vrednost, ki jo sistem izračuna na podlagi izmerjenih časov po različnih formulah. Katera formula se bo uporabila, določi avtor atrd datoteke. Primer: če bosta v atrd datoteki definirana indikatorja

{

"Name" : "Tmin",

"Description" : "Sorting MIN time",

"Type" : "timer",

"Subtype" : "0 MIN"

}

{

"Name" : "Tavg",

"Description" : "Sorting AVG time",

"Type" : "timer",

"Subtype" : "0 AVG"

}

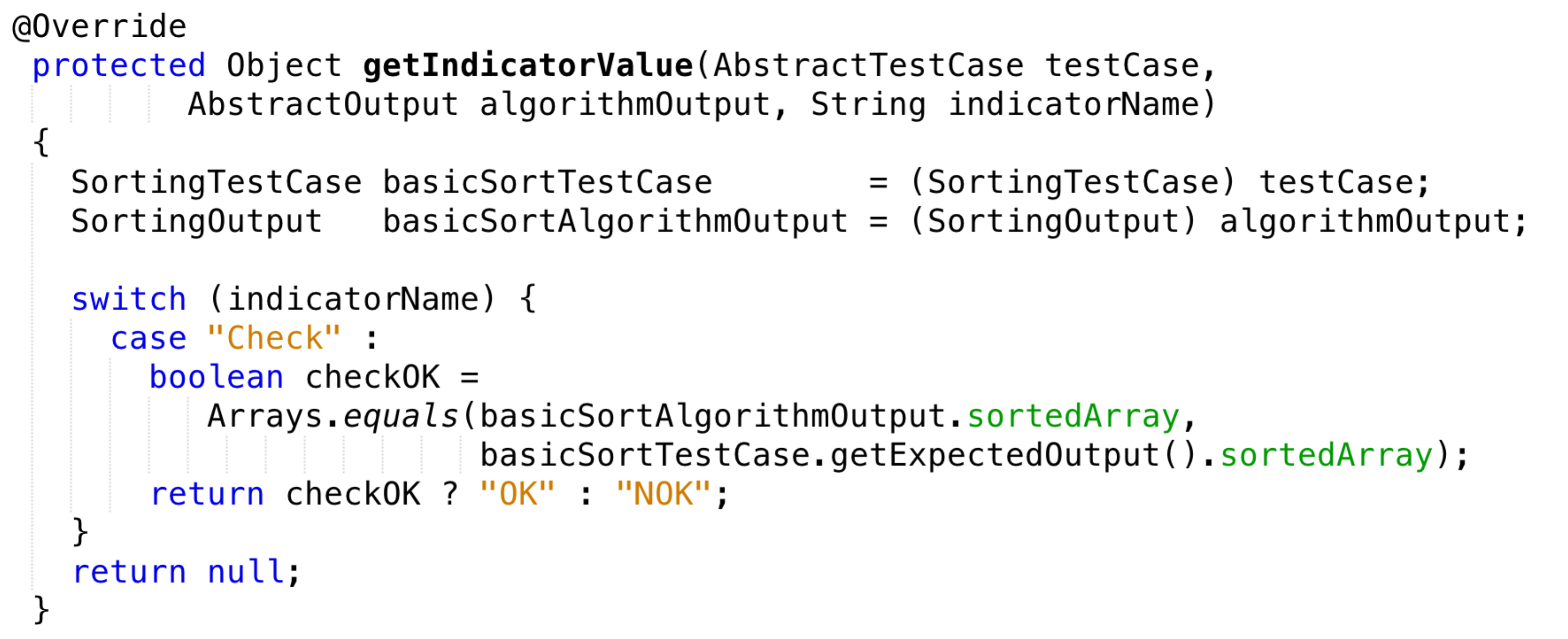
bosta v izhodni datoteki zapisana minimalni in povprečni čas izvajanja algoritma pri TestRepeat ponovitvah tega testa. Osnovni namen ponavljanja je izničenje naželjenih vplivov iz okolja. Pri natančnem ocenjevanju izvajanja algoritma običajno uporabimo minimalno vrednost (pri tem izvajanju je prišlo do najmanj motenj iz okolja).

Vrednost specifičnih indikatorjev se določi v metodi done() metodi na podlagi podatkov o izvajanju algoritma. Metoda done(), ki je že implementirana v razredu AbstractAlgorithm (in je, razen v res posebnih primerih, ni treba spreminjati) za izračun indikatorjev uporabi metodo getIndicatorValue(),

**protected** **Object** getIndicatorValue(AbstractTestCase testCase,

AbstractOutput algorithmOutput, String indicatorName)

ki je definirana v razredu [Project]Output in prejme tri parametre: testni primer, izhod algoritma in ime indikatorja. Ker sta v testnem primeru ”zapakirana” originalni vhod in pričakovan izhod, je naloga metode sorazmerno preprosta: preveriti mora, ali se (oziroma, v kolikšni meri se) dejanski izhod algoritma ujema s pričakovanim izhodom in izračunati vrednost indikatorja. Primer implementacije te metode v razredu SortingOutput je podan spodaj:



### Meritve s števci

Ime atrd datoteke: proj/Project-cnt.atrd

Končnica izhodne datoteke: .cnt

Koda algoritma lahko vsebuje ukaz //@COUNT{counter\_name, value} ki poveča vrednost števca counter\_name za value (primer: //@COUNT{SWAP, 1} poveča vrednost števca SWAP za 1). Po končanem izvajanju izvajalni sistem vrne vrednost vseh števcev, ki so navedeni v polju Indicators v datoteki proj/Project-cnt.atrd. (to polje določa tudi vrstni red zapisa števcev v izhodno datoteko). Polje Parameters v atrd datoteki je pri tej meritvi opcijsko.

Meritev s števci vsak algoritem nad posameznim testom požene samo enkrat. Parameter TestRepeat v tej datoteki se ignorira.

Pred izvajanjem meritve s števci se izvorna koda [alg\_name]Algorithm.java prepiše v datoteko [alg\_name]ALgorithm\_COUNT.java, pri tem pa se vsi ukazi //@COUNT{counter\_name, value} zamenjajo z odgovarjajočimi javanskimi ukazi za povečevanje števcev. Ob izvajanju cnt meritve, se nato izvaja razred [alg\_name]ALgorithm\_COUNT.class.

Pri prepisu datoteke se

1) vse pojavitve

//@COUNT{cnt\_name, value}

nadomestijo z

Counters.addToCounter("cnt\_name", value);

2) vse vrstice, ki vsebujejo //@REMOVE\_LINE, odstranijo.

Primer: Izvorna koda Koda v \_COUNT datoteki

|  |  |
| --- | --- |
| /\*//@REMOVE\_LINE  //@COUNT{CMP, 1}  if (a[e1] != a[e2]) {  //@COUNT{CMP, 1}  }  \*///@REMOVE\_LINE    **if** (a[e1] != a[e2] && a[e2] != a[e3]) {  //@COUNT{CMP, 1}  **while** (a[++less] < pivot1) {  //@COUNT{CMP, 1}  }  } | Counters.addToCounter("CMP", 1);  **if** (a[e1] != a[e2]) {  Counters.addToCounter("CMP", 1);  }  **if** (a[e1] != a[e2] && a[e2] != a[e3]) {  Counters.addToCounter("CMP", 1);  **while** (a[++less] < pivot1) {  Counters.addToCounter("CMP", 1);  }  } |

### Meritve s prirejenim JVM

Ime atrd datoteke: proj/Project-jvm.atrd

Končnica izhodne datoteke: .jvm

Pri tej meritvi štejemo število posameznih bytecode ukazov ter velikost porabljenega pomnilnika (ukazi new, newarray, ...). Za to meritev uporabljamo drugačno izvajalno okolje (prirejen JVM, ki šteje ukaze).

## Izdelava projekta

Skrbnik projekta ustvari nov projekt, algoritem in testno množico z naslednjimi ukazi:

java algator.Admin –cp <ime\_projekta>

java algator.Admin –ca <ime\_projekta> <ime\_algoritma>

java algator.Admin –ct <ime\_projekta> <ime\_testne\_množice>

Pri tem vsi podatki za nov projekt nastanejo v direktoriju $ALGATOR\_DATA\_ROOT/projects/PROJ-<ime\_projekta> in njegovih poddirektorijih. Privzet direktorij lahko skrbnik spremeni s stikalom –dr.

Za pravilno delovanje projekta je treba še:

**1) Testne množice**

* V datoteki [TestSet].atts vpisati pravilne vrednosti za parametre N, TimeLimit, TestRepeat.
* V datoteki [TestSet].txt napisati N vrstic (N testnih primerov)

**2) Izvorna koda projekta**

* V datoteki [Project]TestCase.java definirati podatkovno strukturo
* V datoteki [Project]TestSetIterator.java v metodi getCurrent()
  + na podlagi podatkov iz currentInputLine ustvariti vhodne podatke za izvajanje algoritma na tem testnem primeru in jih zapiše v tCase
  + med parametre testnega primera dodati vse vhodne parametre, ki so zapisani v currentInputLine
* V datoteki [Project]AbsAlgorithm.java
  + v metodi init() vhodne podatke, ki so zapisani v tCase, prepisati v interne atribute in opraviti vse potrebne predpriprave teh podatkov,
  + definirati podpis metode execute(),
  + v metodi run() klicati metodo execute() s prej pripravljenimi podatki
  + v metodi done() na podlagi rezultatov, ki jih generira metoda execute(), izračunati vrednost indikatorjev in jih zapisati v zbirko izhodnih indikatorjev result.

**3) Izvorna koda algorima**

* V datoteki [Algorithm]Algorithm.java napisati metodo execute() (izvajanje algoritma).

**4) Ostali podatki**

Vse zgoraj opisane naloge (1, 2, in 3) mora skrbnik opraviti za pravilno »delovanje« projekta (prevajanje kode in izvajanje algoritmov). Za administrativne potrebe (vzdrževanje sistema, prezentacija rezultatov, analiza, ...) pa mora skrbnik vpisati tudi ostale podatke v konfiguracijskih datotekah (Name, Description, ShortName, ...) in vsebino HTML datotek (te datoteke se uporabljajo za prikaz rezultatov na spletu).

# Konfiguracija sistema

Vsi podatki, s katerimi upravlja sistem ALGator, so zapisani v mapi <algator\_root> in njenih podmapah v tekstovnih konfiguracijskih datotekah tipa JSON ali CSV. Tip posamezne datoteke je razviden iz opisa datoteke v tem poglavju.

Sistem ALGator uporabi vrednosti sistemske spremenljivke $ALGATOR\_ROOT in predpostavi, da kaže na <algator\_root>. Če ta spremenljivka ni nastavljena, ALGator v ta namen uporabi trenutni direktorij. Podatki o projektih se nahajajo na $ALGATOR\_DATA\_ROOT. Če vrednost te sistemske spremenljivke ni nastavljena, se uporabi $ALGATOR\_ROOT/data\_root. Obe vrednosti lahko uporabnik nastavi tudi s pomočjo stikal –r in –dr ob klici programov.

Vsebina datotek, ki predstavljajo konfiguracijo sistema je opisana v datoteki ALGator\_conf.docx.