Leksion 5

Klasat, Objektet, Instancat, Metodat

1. Përcaktimi i klasave dhe krijimi i objekteve

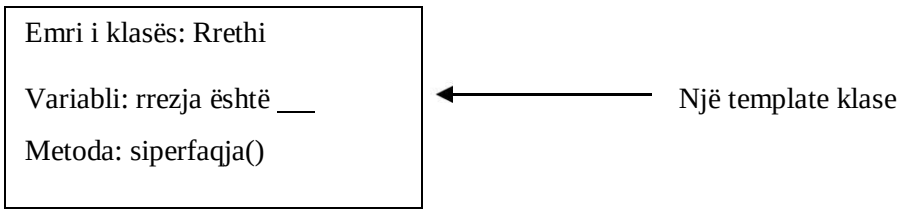
Programimi i orientuar në objekte (OOP) përfaqëson një përpjekje për ti sjellë programet sa më afër mënyrës sesi njerëzit mendojnë dhe përballen me jetën reale. *Objektet* janë entitete në botën reale që mund të identifikohen qartësisht. Ato mbajnë informacione dhe ndërveprojnë me njëra -tjetrën. Për shembull një student, një bankë, një rreth, një buton, qoftë dhe një hua, të gjitha mund të shihen si objekte. Një objekt ka një identitet, gjendje dhe sjellje unike.

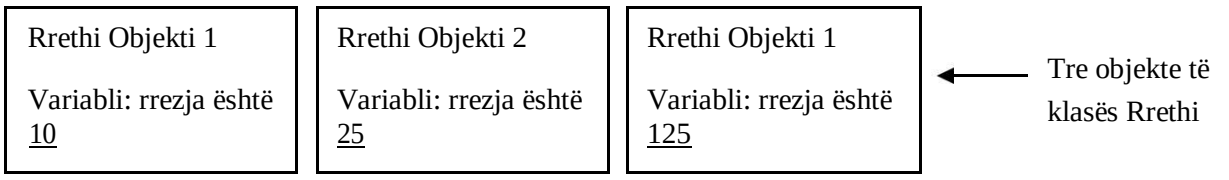
Gjendja e një objekti, gjithashtu e njohur si *vetitë* ose *atributet* e tij, paraqitet nga fushat e të dhënave (që përcaktohen nga variablat) me vlerat e tyre aktuale. Rrethi si objekt, për shembull, ka një variabël, rrezja, i cili është një veti që karakterizon një rreth. Një drejtkëndësh ka dy variabla, gjatësia dhe gjerësia, që janë veti që karakterizojnë një drejtkëndësh.

Sjellja e një objekti, gjithashtu e njohur si *veprimet* e tij, përcaktohet nga metodat. Kur thirret një metodë mbi një objekt, i kërkohet objektit të kryejë një veprim. Për shembull, ju mund të deklaroni një metodë siperfaqja() për objektet rreth. Një objekt rreth, mund të thërrasë metodën siperfaqja() për të kthyer sipërfaqen e tij.

Objektet janë shumë të lidhura me klasat. Ne kemi parë që klasat përmbajnë variabla dhe metoda. Dhe objektet përmbajnë variabla dhe metoda, por nga ndryshojnë ato nga klasat? Zakonisht përdoret termi i përgjithshëm dhe thuhet që objektet "i përkasin" klasave. Nga pikëpamja e programimit është më mirë të thuhet që klasat përdoren për të krijuar objekte. Një *klasë* është një template, një projekt ose kontratë për ndërtimin e objekteve. Pjesët jo-statike të klasës specifikojnë ose shpjegojnë se çfarë do të përmbajnë (dhe bëjnë) variablat dhe metodat e objekteve. Pjesët jo -statike të klasës janë një template për objektet, pasi çdo objekt merr kopjen e tij të pjesëve jo-statike të klasës. Objektet e të njëjtit tip janë përcaktuar duke përdorur një klasë të përbashkët. *Një objekt është një instancë e një klase.* Ju mund të krijoni shumë instanca të një klase. Marrëdhënia midis klasave dhe objekteve është analoge me atë midis një recete të kekut me mollë dhe disa kekëve me mollë. Ju mund të bëni sa kek me mollë të dëshironi nga një recetë e vetme.

Më poshtë tregohet një klasë e quajtur Rrethi dhe tre objektet e saj:





Një klasë përdor *variablat* për të përcaktuar fushat e të dhënave dhe *metodat* për të përcaktuar veprimet. Një klasë ofron metoda të një tipi të veçantë, të cilat thirren për të krijuar një objekt të ri. Këto metoda njihen me emrin *konstruktorë*.

Më poshtë jepet një shembull i përcaktimit të klasës për objektet rreth.

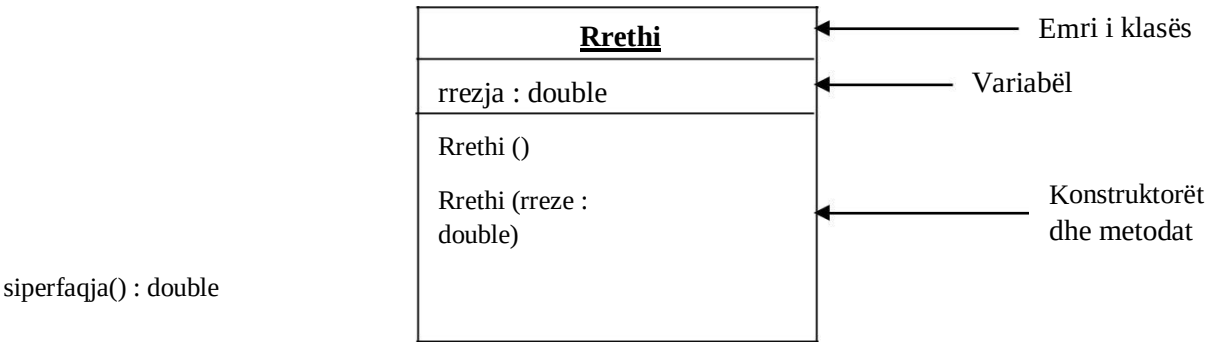
```
class Rrethi {
/** rrezja e ketij rrethi */
double rrezja = 1.0;
/** Nderton nje objekt rreth */
Rrethi() {
}
/** Nderton nje objekt rreth */
Rrethi(double rreze) {
rrezja = rreze;
}
/** kthen siperfaqen e ketij rrethi */
double siperfaqja() {
return rrezja * rrezja * Math.PI;
}
}
```

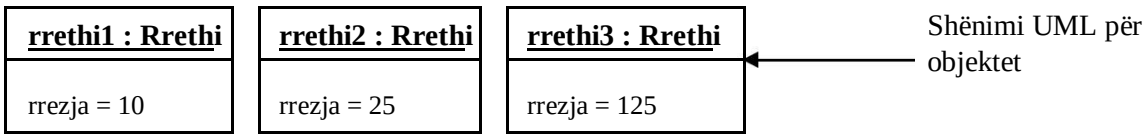
Variabël

Konstruktorë

Metodë

Klasa Rrethi është ndryshe nga të gjitha klasat që kemi parë deri tani. Ajo nuk ka një metodë main, prandaj nuk mund të ekzekutohet; ajo është thjesht një përcaktim i objekteve rreth. Diagrama UML e klasës Rrethi ose thjesht diagrama e klasës Rrethi është paraqitur më poshtë:





Në diagramën e klasës, variabli shënohet si:

emriVariablit : tipiVariablit

Konstruktori shënohet si :

EmriKlasës (emriParametrit : tipiParametrit)

Metoda shënohet si:

emriMetodës(emriParametrit : tipiParametrit) : tipiKthimit

Shembull: Programi i mëposhtëm përcakton klasën Rrethi1 dhe e përdor atë për të krijuar objekte. Programi ndërton tre objekte rreth me rreze 1.0, 25 dhe 125 dhe shfaq rrezen dhe sipërfaqen për secilin prej objekteve. Ndryshon rrezen e objektit të dytë në 100 dhe shfaq rrezen dhe sipërfaqen e tij të re.

```
public class TestRrethi1 {
  public static void main(String[] args) {
  Rrethi1 rrethi1 = new Rrethi1();
  System.out.println("Siperfaqja e rrethit me rreze " + rrethi1.rrezja + " eshte " + rrethi1.siperfaqja());

  Rrethi1 rrethi2 = new Rrethi1(25);
  System.out.println("Siperfaqja e rrethit me rreze " + rrethi2.rrezja + " eshte " + rrethi2.siperfaqja());

  Rrethi1 rrethi3 = new Rrethi1(125);
  System.out.println("Siperfaqja e rrethit me rreze " + rrethi3.rrezja + " eshte " + rrethi3.siperfaqja());

  rrethi2.rrezja = 100;
  System.out.println("Siperfaqja e rrethit me rreze " + rrethi2.rrezja + " eshte
" + rrethi2.siperfaqja());
            }
}

class Rrethi1 {
double rrezja;

  Rrethi1(){
    rrezja = 1.0;
  }

  Rrethi1(double rreze){
    rrezja = rreze;
  }

  double siperfaqja(){
    return rrezja * rrezja * Math.PI;
  }

}
```

Kur programi bëhet run në Eclipse, do të printohet:

```
Siperfaqja e rrethit me rreze 1.0 eshte 3.141592653589793
Siperfaqja e rrethit me rreze 25.0 eshte 3.141592653589793
Siperfaqja e rrethit me rreze 125.0 eshte 3.141592653589793
Siperfaqja e rrethit me rreze 100.0 eshte 31415.926535897932
```

Programi përmban dy klasa. Klasa TestRrethi1 ka si qëllim testimin e klasës Rrethi1. Kur ekzekutohet programi, thirret metoda main e klasës TestRrethi1. Ju mund ti vendosni të dyja klasat në një skedar por vetëm një klasë në skedar mund të jetë publike. Për më tepër klasa publike duhet të ketë të njëjtin emër si emri i skedarit. Në këtë rast emri i skedarit do të ishte TestRrethi1.java meqenëse klasa TestRrethi1 është publike. Klasat mund të vendosen në skedarë të ndryshëm, dhe klasa që ekzekutohet është klasa që përmban metodën main.

Klasa TestRrethi1 përmban metodën main që krijon tre objekte. Variablat rrethi1, rrethi2 dhe rrethi3 janë të tipit Rrethi1. Secili variabël mban një referencë të një objekti. Operatori new përdoret për të krijuar një objekt nga konstruktori. Tre objektet e krijuara kanë të dhëna të ndryshme por metoda të njëjta. Ju mund të gjeni sipërfaqet e tyre përkatëse duke përdorur metodën siperfaqja().

Variablat mund të aksesohen përmes referencës të objektit, duke përdorur rrethi1.rrezja, rrethi2.rrezja dhe rrethi3.rrezja, përkatësisht. Objekti mund të thërrasë metodën e tij përmes referencës të objektit duke përdorur rrethi1.siperfaqja(), rrethi2.siperfaqja() dhe

rrethi3.siperfaqja(), përkatësisht. Tre objektet janë të pavarura. Rrezja e rrethi2 është ndryshuar në 100. Më pas është printuar rrezja dhe sipërfaqja e objektit pas ndryshimit të rrezes.

Programet në Java mund të shkruhen në disa mënyra. Ju mund ti kombinoni dy klasat e programit të mësipërm në një të vetme, si më poshtë:

```
public class Rrethi1 {

public static void main(String[] args) { Rrethi1 rrethi1 = new Rrethi1();
System.out.println("Siperfaqja e rrethit me rreze " + rrethi1.rrezja + " eshte + rrethi1.siperfaqja());

Rrethi1 rrethi2 = new Rrethi1(25);
System.out.println("Siperfaqja e rrethit me rreze " + rrethi2.rrezja + " eshte " + rrethi2.siperfaqja());

Rrethi1 rrethi3 = new Rrethi1(125);
System.out.println("Siperfaqja e rrethit me rreze " + rrethi3.rrezja + " eshte " + rrethi3.siperfaqja());

rrethi2.rrezja = 100;
System.out.println("Siperfaqja e rrethit me rreze " + rrethi2.rrezja + " eshte " + rrethi2.siperfaqja());
}

double rrezja;

Rrethi1(){
rrezja = 1.0;
}

Rrethi1(double rreze){
rrezja = rreze;
}

double siperfaqja(){
return rrezja * rrezja * Math.PI;
}
}
```

Meqenëse klasa e kombinuar e përmban metodën main, ajo mund të ekzekutohet nga interpretuesi i Java.

1. Krijimi i objekteve duke përdorur konstruktorët

Konstruktorët janë një tip i veçantë metode. Në fakt konstruktorët janë metoda që krijojnë objekte. Për kompjuterin, procesi i ndërtimit të një objekti do të thotë, fillimisht gjetja e kujtesës të pa përdorur në stivë e cila mund të përdoret për të mbajtur objektin, dhe më pas mbushja me variablat e instancës të objektit. Konstruktorët nuk janë metoda instance, sepse ato nuk i përkasin objekteve. Meqenëse konstruktorët janë përgjegjës për krijimin e objekteve, ata ekzistojnë përpara krijimit të ndonjë objekti. Konstruktorët kanë tre veçori:

- Një konstruktor duhet të ketë emër të njëjtë me vetë klasën ku është përcaktuar.
- Konstruktorët nuk kanë një tip kthimi, madje as *void*.
- Konstruktorët thirren duke përdorur operatorin new kur krijohet një objekt. Konstruktorët luajnë rolin e inicializimit të objekteve.

Modifikuesit që mund të përdoren në deklarimin e një konstruktori janë modifikuesit e aksesit public, private dhe protected. Një konstruktor nuk mund të deklarohet si static. Një konstruktor ka një trup

metode të formës të zakonshme, një bllok statement-esh. Kur përcaktohet një konstruktor, ai mund të ketë disa parametra formalë, ose mund edhe të mos ketë asnjë parametër.

Është gabim të përdoret fjala çelës void përpara një konstruktori. Për shembull:

```
public void Rrethi(){
}
```

Në këtë rast Rrethi()është një metodë dhe jo një konstruktor. Për të ndërtuar një objekt nga një klasë, thirret një konstruktor i klasës duke përdorur operatorin new, si më poshtë:

new <EmriKlasës>(<lista_e_argumentave>);

Kjo shprehje alokon kujtesë për objektin, inicializon variablat e instancës të objektit, dhe kthen një referencë të objektit. Kjo referencë është vlera e shprehjes, dhe ajo vlerë mund të ruhet me një statement caktimi në një variabël, kështu pas ekzekutimit të statement-it të caktimit variabli i referohet objektit të ri të krijuar. Lista e argumentave mund të jetë edhe boshe.

Për shembull:

```
new Rrethi();
```

krijon një objekt të klasës Rrethi duke përdorur konstruktorin e parë të përcaktuar, dhe

```
new Rrethi(25);
```

krijon një objekt duke përdorur konstruktorin e dytë të përcaktuar në klasën Rrethi.

Zakonisht klasat kanë të paktën një konstruktor. Megjithatë një klasë mund të përcaktohet edhe pa konstruktorë. Në këtë rast sistemi do të ofrojë një *konstruktor default*, të cilin e ndërton kompilatori i Java. Konstruktori default alokon kujtesë dhe inicializon variablat e instancës.

2. Aksesimi i objekteve përmes variablave të referencës

Objektet e krijuara janë alokuar në kujtesë. Ato mund të aksesohen nëpërmjet variablave të referencës.

3.1 Variablat e referencës dhe tipet e referencës

Në Java, një klasë është një tip, e ngjashme me tipet primitive si int dhe boolean. Kështu, emri i një klase mund të përdoret për të specifikuar tipin e një variabli në një statement deklarimi, ose tipin e një parametri formal ose tipin e kthimit të një funksioni. Objektet aksesohen nëpërmjet variablave të referencës të objekteve, të cilat përmbajnë referenca të objekteve. Këta variabla deklarohen duke përdorur sintaksën e mëposhtme:

<EmriKlasës> <varRefObjektit>;

Një klasë është një tip reference, që do të thotë që një variabël i një tipi klase mund të referojë ose shënjojë një instancë të një klase. Statement-i i mëposhtëm deklaron variablin rrethiX të tipit Rrethi:

Rrethi rrethiX;

Megjithatë deklarimi i një variabli nuk krijon një objekt. Në Java asnjë variabël nuk mund të mbajë një objekt. Një *variabël mund vetëm të mbajë një referencë të një objekti*. Variabli mban informacionin e nevojshëm për të gjetur një objekt në kujtesë. Ky informacion quhet një *referencë* ose *shënjues* i objektit. Ekziston një porcion specifik në kujtesë i quajtur *stivë* ku jetojnë objektet. Një referencë e një objekti është adresa e lokacionit të kujtesës ku ruhet objekti. Kur ju përdorni një variabël referencial, kompjuteri përdor referencën në variabël për të gjetur objektin aktual. Në një program objektet krijohen duke përdorur një operator të quajtur new, që krijon një objekt dhe kthen një referencë për atë objekt. (Në fakt operatori new thërret një metodë të veçantë të quajtur një konstruktor në klasë.) Për shembull, duke supozuar se rrethiX është një variabël i tipit Rrethi, i deklaruar si më sipër, statementi:

rrethiX = new Rrethi();

do të krijojë një objekt të ri që është një instancë e klasës Rrethi, dhe ai do të ruajë një referencë të atij objekti në variablin rrethiX. Vlera e variablit është një referencë ose shënjues për objektin. Vetë objekti është diku në stivë. Në këtë rast thuhet që variabli rrethiX referon ose shënjon tek një objekt që është një instancë e klasës Rrethi.

Ne mund të shkruajmë një statement të vetëm që kombinon deklarimin e një variabli referencial të objektit, krijimin e një objekti dhe caktimin e një reference të objektit tek variabli duke përdorur sintaksën e mëposhtme:

<EmriKlasës> <varRefObjektit> = new <EmriKlasës>();

Për shembull:

Rrethi rrethiX = new Rrethi();

Variabli rrethiX mban një referencë të një objekti Rrethi.

Tabelat trajtohen si objekte në Java, ato krijohen duke përdorur operatorin new. Një variabël i një tabele është një variabël që mban një referencë të një tabele (array).

3.2 Aksesimi i të dhënave dhe i metodave të objekteve

Pas krijimit të një objekti, të dhënat e tij mund të aksesohen dhe metodat e tij mund të thirren duke përdorur operatorin pikë (.), të njohur dhe si operatori i aksesimit të anëtarëve të objektit:

<varRefObjektit>.<variabli> i referohet një variabli në objekt.

<varRefObjektit>.<metoda>(<argumentat>) thërret një metodë mbi objektin.

Për shembull:

rrethiX.rrezja i referohet variablit rrezja në objektin rrethiX.

rrethiX.sipërfaqja() thërret metodën sipërfaqja mbi rrethiX.

Një objekt që krijohet duke përdorur një klasë thuhet të jetë instancë e asaj klase. Variablat që përmban objekti quhen *variabla instance*, kurse metodat që përmban objekti quhen *metoda instance*. Variabli rrezja është një variabël instance, sepse ajo është e varur mbi një instancë specifike. Për të njëjtën arsye, metoda sipërfaqja është një metodë instance sepse ju mund ta thërrisni atë vetëm mbi një instancë specifike.

3.3 Variablat e tipit referencial dhe vlerat *null*

Variablat mund të jenë të tipeve referencial. Për shembull, klasa e mëposhtme Student përmban një variabël emri të tipit String. String është një klasë e paracaktuar në Java.

```
public class Student {
    String emri; //emri ka vleren default null int mosha; //mosha ka
    vleren default 0
    Boolean eshteDegaTI; // eshteDegaTI ka vleren default false char gjinia; //gjinia ka
    vleren default '\u0000'
}
```

Nëse një variabël i tipit referencial nuk referon mbi ndonjë objekt, ai mban një vlerë Java të veçantë, *null*. Vlera default e një variabli është null për një tip referencial si String, 0 për një tip int, false për një tip boolean, dhe '\u0000' për një tip char. Gjithsesi, Java nuk cakton ndonjë vlerë default për një variabël lokal brenda një metode. Kodi i mëposhtëm shfaq vlerat e variablave emri, mosha, eshteDegaIT dhe gjinia për një objekt të tipit Student:

```
public class Test {
    public static void main(String[] args) {
    Student student = new Student();
    System.out.println("emri? " + student.emri);
    System.out.println("mosha? " +  student.mosha);
    System.out.println("DegaInformatike? " + student.eshteDegaTI);
    System.out.println("gjinia? " + student.gjinia);
    }
}
```

Java nuk jep një vlerë default për variablat lokalë brenda metodave. Kodi i mëposhtëm ka një error kompilimi sepse variablat lokalë x dhe y nuk janë inicializuar:

```
public class Test {
    public static void main(String[] args) {
        int x; // x nuk ka vlere default
        String y; // y nuk ka vlere default
        System.out.println("x eshte " + x);
        System.out.println("y eshte " + y);
    }
}
```

Ekziston mundësia që një variabël si student, tipi i të cilit jepet nga një klasë, të mos referojë tek ndonjë objekt. Në këtë rast thuhet që student mban një shënjues null ose referencë null. Shënjuesi null në Java shkruhet si "null". Ju mund të ruani një referencë null në variablin student duke shkruar:

```
Student student;

student = null;
```

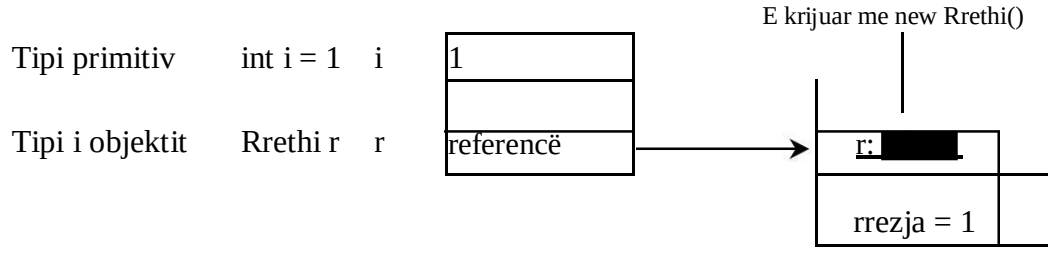
null është vlera aktuale e ruajtur në variabël dhe jo një shënjues e diçkaje tjetër. Nuk është korrekte të thuhet që variabli "shënjon tek null" sepse në fakt variabli është null. Ju mund ta testoni nëse vlera e variablit është null:

```
if(std == null)...
```

Kur vlera e variablit është null, nuk lejohet ti referohemi variablave të instancës ose metodave të instancës përmes këtij variabli, pasi ai nuk referon tek ndonjë objekt.

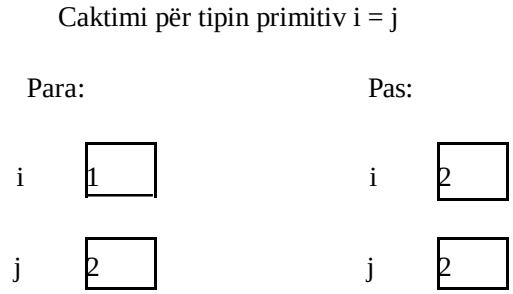
3.4 Dallimi midis variablave të tipeve primitive dhe variablave të tipit referencial

Çdo variabël përfaqëson një lokacion një kujtesë që mban një vlerë. Kur ju deklaroni një variabël, ju i tregoni kompilatorit tipin e vlerës që mund të mbajë variabli. Për një variabël të një tipi primitiv, vlera është e tipit primitiv. Për një variabël të tipit referencial, vlera është një referencë e lokacionit të objektit në kujtesë. Në figurën e mëposhtme, vlera e variablit i të tipit int është vlera 1, kurse vlera e variablit referencial r mban një referencë të përmbajtjes të objektit Rrethi në kujtesë.

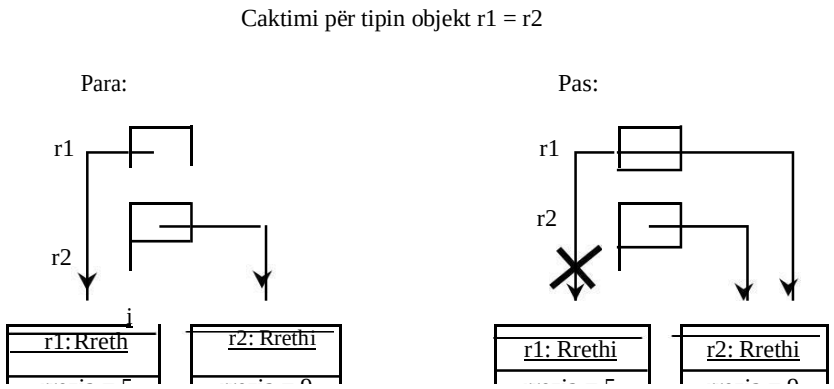


Kur ju caktoni një variabël tek një variabël tjetër, variabli tjetër vendoset me të njëjtën vlerë. Për një variabël të një tipi primitiv, vlera reale e një variabli caktohet tek variabli tjetër. Për një variabël të një tipi referencial, referenca e një variabli caktohet tek variabli tjetër. Për shembull, statement-i i caktimit i = j;

kopjon përmbajtjen e j tek i për variablat primitivë. Fillimisht variabli i mban vlerë 1 dhe j mban vlerën 2. Pas statement-it të caktimit të dy variablat kanë të njëjtën vlerë:



Statement-i i caktimit r1 = r2 kopjon referencën e variablit të referencës r2 tek variabli referencial r1. Pas caktimit, të dy variablat i referohen të njëjtit objekt:



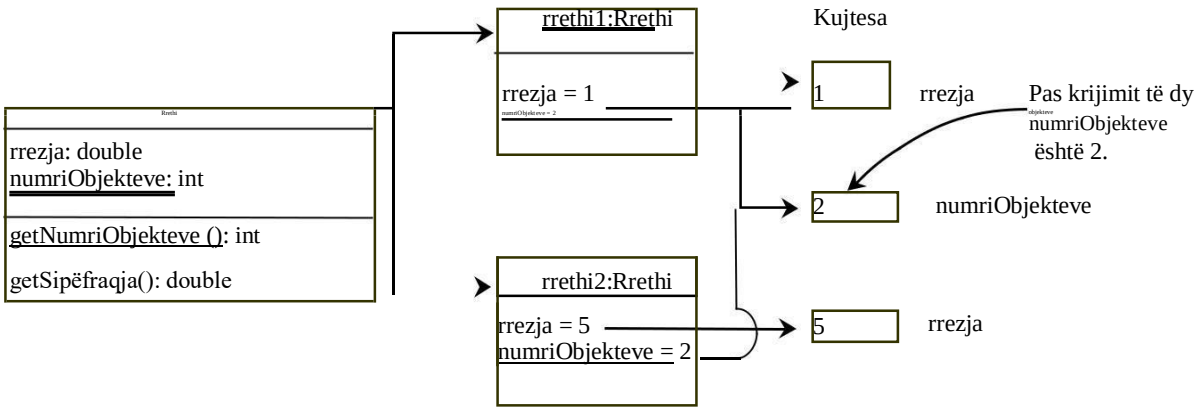
Tashmë objekti të cilit i referohej variabli r1 është i pa dobishëm dhe prandaj njihet si *garbage (mbeturina)*. Hapësira që ka zënë ky objekt në kujtesë do të lirohet. Ky proces quhet *garbage collection (mbledhje e mbeturinave)*. Nëse ju e dini që një objekt nuk nevojitet më, ju mund të caktoni në mënyrë eksplicite null tek një variabël referencial për objektin. Java Virtual Machine do të marrë hapësirën nëse objekti nuk referohet nga një variabël referencial.

3. Variablat dhe metodat statike

Variabli rrezja tek klasa Rrethi është një *variabël instance*. Variabli rrezja për dy objekte të ndryshme ruhet ne lokacione të ndryshme në kujtesë. Kështu rrethi1.rrezja është i pavaruar nga rrethi2.rrezja. Ndryshimet e bëra tek rrezja e njërit objekt nuk ndikojnë tek rrezja e objektit tjetër.

Variablat statikë ose siç njihen ndryshe si *variablat e klasës* i ruajnë vlerat e variablave në një lokacion të përbashkët në kujtesë. Nëse një nga objektet ndryshon vlerën e një variabli statik, do të preken të gjithë objektet e të njëjtës klasë. Java suporton variablat dhe metodat statike. Metodat statike mund të thirren pa krijuar një instancë të një klase.

Le të modifikojmë klasën Rrethi duke shtuar një variabël statik numriObjekteve që numëron numrin e objekteve rreth të krijuara. Kur krijohet objekti i parë i kësaj klase, numriObjekteve është 1. Kur krijohet objekti i dytë, numriObjekteve bëhet 2. Diagrama UML e klasës të re rreth tregohet më poshtë. Klasa Rreth përcakton variablin e instancës rrezja dhe variablin statik numriObjekteve, metodat e instancës getRrezja, setRrezja, dhe getSipërfaqja, dhe metodën statike getNumriObjekteve. Variablat dhe metodat statike janë të nënvizuara në diagramën UML të klasës.



Për të deklaruar një variabël statik ose për të përcaktuar një metodë statike duhet të vendosni modifikuesin static në deklarimin e variablit ose metodës. Variabli statik numriObjekteve dhe metoda statike getNumriObjekteve mund të deklarohen si më poshtë:

```
static int numriObjekteve;
static int getNumriObjekteve() { return numriObjekteve;
}
```

Le të shohim tani klasën e re Rrethi2 :

```
public class Rrethi2 {

double rrezja; //variabël instance
static int numriObjekteve = 0; //variabël statik

Rrethi2(){
rrezja = 1.0;
numriObjekteve ++;
}

Rrethi2(double rreze){
rrezja = rreze;
numriObjekteve ++;
}

static int getNumriObjekteve(){  //metodë statike
return numriObjekteve;
}

double getSipërfaqja(){ //metodë instance
return rrezja * rrezja
* Math.PI;
}
}
```

Metodat dhe variablat e instancës i takojnë instancave dhe mund të përdoren vetëm pasi të jenë krijuar instancat. Ato mund të aksesohen nëpërmjet një variabli referencial. Metodat dhe variablat statikë mund të aksesohen nga një variabël referencial ose nga emri i klasës të tyre. Programi i mëposhtëm demostron mënyrën sesi përdoren variablat dhe metodat e instancës dhe ato statike, si dhe ilustron efektet e përdorimit të tyre:

```
public class TestRrethi2 {

public static void main(String[] args) {
System.out.println("Përpara krijimt të objekteve");
System.out.println("Numri i objekteve rreth është: "+ Rrethi2.numriObjekteve);
Rrethi2 rr1= new Rrethi2();

System.out.println("Pasi krijohet rr1");
System.out.println("rr1: rrezja (" + rr1.rrezja +") dhe numri i objekteve rreth (" + rr1.numriObjekteve + ")");
Rrethi2 rr2 = new Rrethi2(5);
rr1.rrezja = 9;

System.out.println("Pasi krijohet rr2 dhe modifikohet rr1");
System.out.println("rr1: rrezja (" + rr1.rrezja + ") dhe numri i objekteve rreth (" + rr1.numriObjekteve + ")");
System.out.println("rr2: rrezja (" + rr2.rrezja + ") dhe numri i objekteve rreth (" + rr2.numriObjekteve + ")");

}
}
```

Ky program afishon:

```
Përpara krijimt të objekteve
Numri i objekteve rreth është: 0
Pasi krijohet rr1
rr1: rrezja (1.0) dhe numri i objekteve rreth (1)
Pasi krijohet rr2 dhe modifikohet rr1
rr1: rrezja (9.0) dhe numri i objekteve rreth (2)
rr2: rrezja (5.0) dhe numri i objekteve rreth (2)
```

Variablat dhe metodat statike mund të përdoren nga metodat statike ose nga metodat e instancës në klasë. Kurse variablat dhe metodat e instancës mund të përdoren vetëm nga metodat e instancës dhe jo nga metodat statike, sepse variablat dhe metodat statike nuk i përkasin ndonjë objekti të veçantë. Prandaj kodi i mëposhtëm është gabim:

```
public class Objekt {
    int i = 5;
    static int k = 2;

    public static void main(String[] args) {
        int j = i; // gabim sepse i është një variabël instance

        metoda1(); //gabim sepse metoda1() është një metodë instance

    }
    public void metoda1(){
        i = i + k + metoda2(i,k);
        //Korrekte, meqenëse variablat dhe metodat statike dhe
        //ato të instancës mund të përdoren në një metodë të instancës.
    }
    public static int metoda2(){
        return (int)(Math.pow(i,j));
    }
}
```

Një pjesë e kodit që është gabim mund të modifikohet duke përftuar kështu një program të saktë:

```
public class Objekt {
    int i = 5;
    static int k = 2;

    public static void main(String[] args) {
        Objekt obj = new Objekt();
        int j = obj.i; //obj.i akseson variablin e instancës të obektit
        obj.metoda1();  //obj.metoda1 thërret metodën e instancës të objektit
        }
    public void metoda1(){
        i = i + k + metoda2(i,k);
    }

    public static int metoda2(){
        return (int)(Math.pow(i,j));
    }
}
```

Por si e vendosni ju nëse një variabël ose metodë do jetë instance ose statike?

Një variabël ose metodë që varet mbi një instancë specifike të klasës duhet të jetë një variabël instance ose metodë instance.

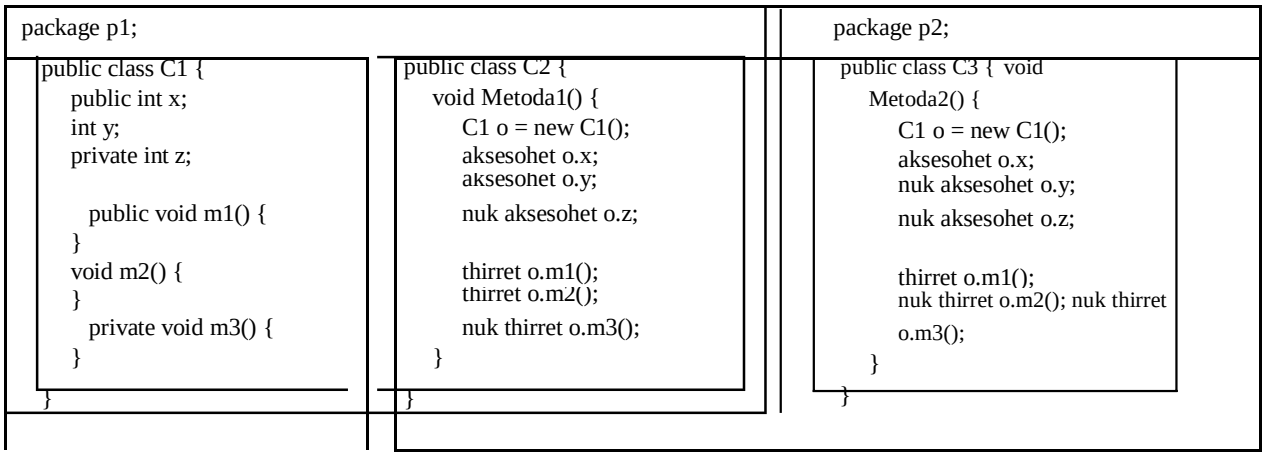
Një variabël ose metodë që nuk varet mbi një instancë specifike të klasës duhet të jetë një variabël statik ose metodë statike.

Për shembull, çdo rreth ka rrezen e tij. Rrezja varet mbi një rreth specifik. Prandaj rrezja është një variabël instance e klasës Rrethi. Meqenëse metoda getSipërfaqja varet mbi një rreth specifik, ajo është një metodë instance. Metoda main është statike dhe mund të thirret direkt nga një klasë.

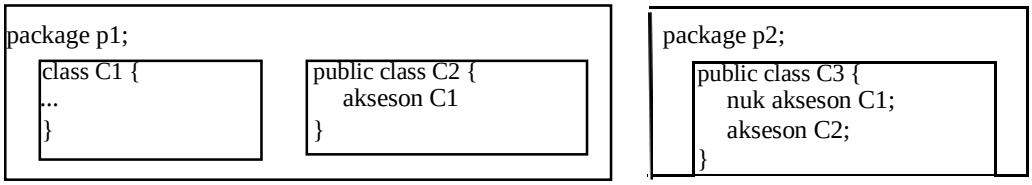
4. Modifikuesit e dukshmërisë (visibility)

Ju mund të përdorni modifikuesin e dukshmërisë public për klasat, metodat dhe fushat e të dhënave për të treguar se ato mund të aksesohen nga çdo klasë tjetër. Nëse nuk është përdorur ndonjë modifikues dukshmërie, atëherë vetë (by default) klasat, metodat dhe fushat e të dhënave mund të aksesohen nga çdo klasë në të njëjtën paketë. Modifikuesi private i bën metodat dhe fushat e të dhënave të aksesueshme vetëm brenda klasës të tyre.

Figura e mëposhtme ilustron sesi aksesohen fushat e të dhënave dhe metodat publike, default dhe private të klasës C1 nga një klasë C2 brenda të njëjtës paketë dhe nga një klasë C3 në një paketë tjetër.



Nëse një klasë nuk është përcaktuar publike, ajo mund të aksesohet vetëm brenda të njëjtës paketë. Për shembull, klasa C1 mund të aksesohet nga C2 por jo nga C3:



Një modifikues vizibiliteti specifikon sesi mund të aksesohet fushat e të dhënave dhe metodat e një klase nga jashtë klasës. Nuk ka kufizime në aksesimin e fushave të të dhënave dhe metodave brenda klasës. Në figurën e mëposhtme, në kodin majtas objekti foo mund të asksesojë anëtarët privatë të klasës Foo pasi është përcaktuar brenda klasës. Kurse në kodin në anën e djathtë objekti foo, nuk mund të aksesojë anëtarët privatë të klasës Foo, pasi ai është përcaktuar në klasën Test dhe jo brenda klasës Foo.

```
public class Foo {
    private boolean x;

    public static void main(String[] args) {
        Foo foo = new Foo();
        System.out.println(foo.x);
        System.out.println(foo.konverto());
    }

    private int konverto(boolean b) { return x ? 1 : -1;
    }
}
```

(a) kjo eshte OK, pasi objekti foo eshte brenda klases Foo

```
public class Test {
    public static void main(String[] args) {
        Foo foo = new Foo();
        System.out.println(foo.x);
        System.out.println(foo.konverto(foo.x));
    }
}
```

(b) gabim, mbasi x dhe konverto janë private në Foo.

Modifikuesi private aplikohet vetëm tek anëtarët e klasës. Modifikuesi public mund të aplikohet tek një klasë ose tek anëtarët e një klase. ***Përdorimi i modifikuesve public dhe private tek variablat lokalë do të shkaktojë një gabim kompilimi.*** Në shumicën e rasteve, konstruktori duhet të jetë public. Gjithsesi, nëse ju doni që ti ndaloni përdoruesit nga krijimi i një instance të një klase, përdorni një konstruktor privat. Për shembull, nuk ka arsye për të krijuar një instancë nga klasa Math, sepse të gjitha fushat e të dhënave dhe metodat janë statike. Për të ndaluar përdoruesit të krijojnë objekte nga klasa Math, konstruktori në java.lang.Math është përcaktuar si më poshtë:

```
private Math() {
}
```

Modifikuesi protected përdoret për të lejuar klasat që trashëgojnë nga një klasë të aksesojnë fushat e të dhënave ose metodat e asaj klase.

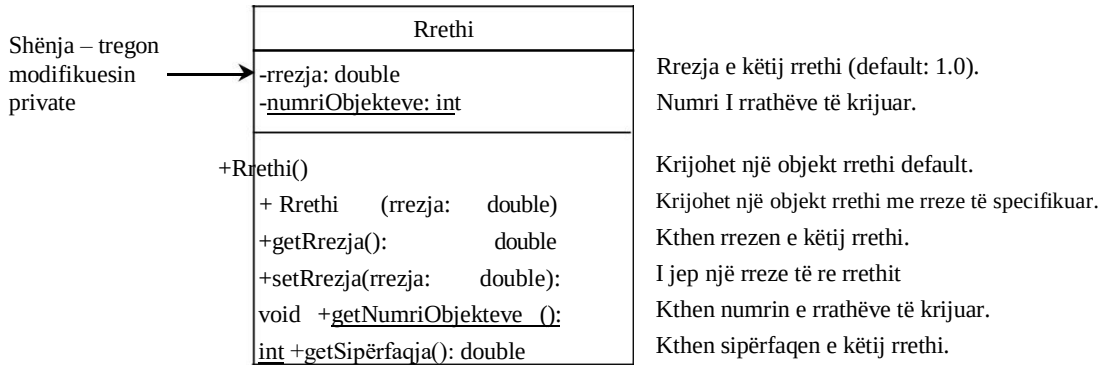
5. Enkapsulimi i fushave të të dhënave

Fushat e të dhënave rrezja dhe numriObjekteve në klasën Rrethi2 mund të modifikohen direkt, p.sh.
rr1.rrezja = 5
ose
rr1.numriObjekteve = 10.
Kjo nuk është një gjë e mirë, për dy arsye:
Së pari, të dhënat mund të ngatërrohen. Për shembull, numriObjekteve përdoret për të numëruar numrin e objekteve të krijuara, por mundet që gabimisht ti jepet një vlerë (p.sh. rr1.numriObjekteve = 10).
Së dyti, klasa bëhet e vështirë për tu mirëmbajtur dhe e pambrojtur ndaj gabimeve. Supozojmë se ju doni të modifikoni klasën Rrethi2 për tu siguruar që rrezja nuk është negative pasi programe të tjera të kenë

përdorur klasën. Ju duhet të ndryshoni jo vetëm klasën Rrethi2 por dhe programet që e përdorin atë, sepse klientët mund ta kenë modifikuar rrezen direkt (p.sh. rr1.rrezja = -5).

Për të penguar modifikimin e fushave të të dhënave, ju duhet të deklaroni fushat e të dhënave si private duke përdorur modifikuesin private. **Kjo njihet si *enkapsulimi i fushave të të dhënave***.
Një fushë të dhëne private nuk mund të aksesohet nga një objekt nga jashtë klasës që përcakton fushën private. Por shpesh një klienti i nevojitet të aksesojë dhe të modifikojë një fushë të dhëne. Për të bërë një fushë të dhëne private të aksesueshme, ofrohet një metodë *get* që kthen vlerën e saj. Për të bërë të mundur modifikimin e një fushe të dhëne private, ofrohet një metodë *set* për ti caktuar një vlerë të re. Një metode *get* i referohemi si një marrës (getter) ose accessor kurse një metode set i referohemi si një caktues (setter) ose mutator.

Le të krijojmë një klasë të re rreth me një fushë të dhene rrezja private dhe metodat e tij get dhe set.
Diagrama e klasës jepet më poshtë:



Klasa e re rreth, e quajtur Rrethi3 përcaktohet si më poshtë:

```
public class Rrethi3 {

    private double rrezja = 1.0; //rrezja e rrethit
    private static int numriObjekteve = 0; //numri i objekteve të krijuara

    Rrethi3(){
        numriObjekteve ++;
    }//ndërton një rreth me rreze 1.0

    Rrethi3(double rreze){
        rrezja = rreze;
        numriObjekteve ++;
    } //ndërton një rreth me rreze të specifikuar

    public double getRrezja(){
          return rrezja;
    } //Kthen rrezen

    public double setRrezja(double rreze){
          rrezja = (rreze >= 0)? rreze : 0;
    } //Cakton një rreze

    public static int getNumriObjekteve(){
          return numriObjekteve;
    }//kthen numrin e objekteve

    public double getSipërfaqja(){
      return rrezja * rrezja * Math.PI;
    } //kthen sipërfaqen e këtij rrethi
}
```

Metoda getRrezja kthen rrezen, dhe metoda setRrrezja(rreze) cakton një rreze të re në objekt. Nëse rrezja është negative, 0 vendoset si rrezja e objektit. Meqenëse këto metoda janë mënyrat e vetme për leximin dhe modifikimin e rrezeve, ju keni kontroll të plotë mbi aksesimin e variablit rrezja. Nëse ju ndryshoni implementimin e këtyre metodave, nuk nevojitet të ndryshoni programet klient. Kjo e bën klasën të lehtë për tu mirëmbajtur. Më poshtë jepet një program klient që përdor klasën Rrethi3 për të krijuar një objekt rreth dhe modifikon rrezen duke përdorur metodën setRrezja.

```
public class TestRrethi3 {

    public static void main(String[] args) {

        // Krijon një rreth me rreze 5.0

        Rrethi3 rrethi1 = new Rrethi3(5.0); System.out.println("Sipërfaqja e rrethit me rreze " +
        rrethi1. getRrezja() + "është " + rrethi1.getSipërfaqja() );

        //rrit rrezen e rrethit me 10%

        rrethi1.setRrezja(rrethi1.getRrezja * 1.1);

        System.out.println("Sipërfaqja e rrethit me rreze " + rrethi1. getRrezja() + "është " +
        rrethi1.getSipërfaqja() );

        System.out.println("Numri i objekteve të krijuara është " + Rrethi3.getNumriObjekteve());
    }
}
```

Fusha e të dhënës rrezja është deklaruar si private. Të dhënat private mund të aksesohen vetëm brenda klasës ku ato janë përcaktuar. Ju nuk mund të përdorni rrethi1.rrezja në programin klient. Një gabim kompilimi do të ndodhë nëse ju do të përpiqeni të aksesoni të dhëna private nga një klient. Meqenëse numriObjekteve është privat, ai nuk mund të modifikohet. Kjo parandalon ngatërrimin.

6. Kalimi i objekteve në metoda

Ju mund të kaloni objekte në metoda. Ashtu si kalimi i një tabele në metoda, kalimi i një objekti është në fakt kalimi i referencës të objektit. Kodi i mëposhtëm kalon objektin rrethiX si një argument tek metoda printoRrethin:

```
public class Test {
    public static void main(String[] args){
        Rrethi3 rrethiX = new Rrethi3(5.0);
        printoRrethin(rrethiX);
    }
    public static void printoRrethin(Rrethi3 rr){ System.out.println("Sipërfaqja e rrethit
        me rreze " + rr.getRrezja() + "është " + rr.getSipërfaqja() );

    }
}
```

Java përdor mënyrën e kalimin me vlerë për argumentat. Në kodin e mësipërm vlera e rrethiX është kaluar tek metoda printoRrethin. Kjo vlerë është një referencë tek një objekt rreth.

7. Tabelat e objekteve

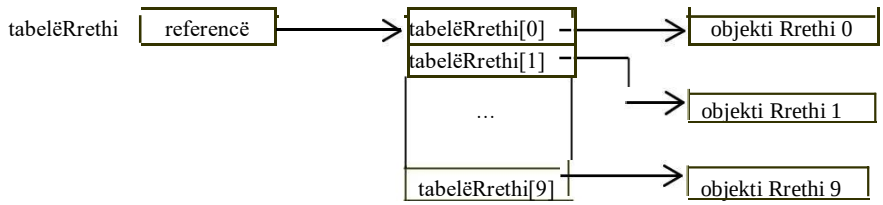
Deri tani kemi parë krijimin e tabelave (arrays) të elementeve të tipit primitiv. Por, mund të krijohen dhe tabela objektesh. Për shembull, statement-i i mëposhtëm deklaron dhe krijon një tabelë prej 10 objektesh Rreth:

Rrethi[] tabelëRrethi = new Rrethi[10];

Për të inicializuar tabelëRrethi, ju mund të përdorni një cikël for:

```
for (int i = 0; i < tabelëRrethi.length; i++) {
   tabelëRrethi[i] = new Rrethi();
}
```

Një tabelë objektesh është një *tabelë e variablave të referencës*. Kështu thirrja e tabelëRrethi[1].getSipërfaqja()pëfshin dy nivele referencimi:



tabelëRrethi i referohet të gjithë tabelës. tabelëRrethi[1] i referohet një objekti Rrethi.

Kur krijohet një tabelë objektesh duke përdorur operatorin new, çdo element në tabelë është një variabël reference me një vlerë default null.

8. Klasat dhe objektet e përhershme (të pandryshueshme)

Normalisht, ju mund të krijoni një objekt dhe të lejoni që përmbajtja e tij të ndryshohet më vonë. Rastësisht, mund të nevojitet krijimi i një objekti, përmbajtja e të cilit nuk mund të ndryshohet pasi ai është krijuar. Ky quhet një objekt i përhershëm dhe klasa e tij quhet klasë e përhershme. Klasa String, për shembull, është e pandryshueshme. Nëse do fshini metodën set tek klasa Rrethi3 që kemi parë më sipër, klasa do ishte e përhershme, sepse rrezja është private dhe nuk mund të ndryshohet pa një metodë set. Nëse një klasë është e pandryshueshme, atëherë gjithë fushat e të dhënave të saj duhet të jenë private dhe ajo nuk mund të përmbajë metoda set publike për asnjërën prej fushave të të dhënave. Një klasë me fusha të dhënash private dhe pa mutator-ë nuk është detyrimisht e pandryshueshme. Që një klasë të jetë e pandryshueshme duhet të plotësojë tre kushte:

- Gjithë fushat e të dhënave të jenë private.
- Mos përmbajë metoda set.
- Mos përmbajë metoda get që kthejnë një referencë të një fushe të dhëne që është e ndryshueshme.

9. Variablat e klasës

Variablave të instancës ose variablave statikë iu referohemi si variablat e klasës ose fushat e të dhënave. Një variabël i përcaktuar brenda një metode njihet si variabël lokal. Variablat e klasës dhe të metodave mund të vendosen në çdo mënyrë në një klasë, siç tregohet më poshtë:

```
public class Rrethi {
        public double gjejSipërfaqen(){
                return rrezja * rrezja * Math.PI;
        }
```

```
        private double rrezja = 1;

}
```

Variabli rrezja dhe metoda gjejSipërfaqen mund të vendosen në renditje të çfarëdoshme.

Vetëm në rastin kur një variabël inicializohet bazuar mbi një referencë tek një variabël tjetër , ky i fundit duhet të jetë deklaruar i pari. Për shembull, variabli i duhet të deklarohet përpara j sepse vlera fillestare e j varet nga i:

```
public class Variabla {
        private int i;
        private int j = i + 1;
}
```

Ju mund ta deklaroni variablin e një klase vetëm një herë, por ju mund të deklaroni të njëjtin emër variabli në një metodë disa herë në blloqe të ndryshme që nuk përfshihen tek njëra-tjetra. Nëse një variabël lokal ka emër të njëjtë me një variabël të klasës, variabli lokal ka prioritet dhe variabli i klasës me të njëjtin emër është i fshehur (hidden). Për shembull, në programin e mëposhtëm x është përcaktuar si një variabël instance dhe si një variabël lokal në metodë:

```
public class Variablat {
    private int x = 0;// variabël instance
    private int y = 0;

    public Variablat() {
    }
    public void m() {
      int x = 1; // variabël lokal
      System.out.println("x = " + x);
      System.out.println("y = " + y);
    }
}
```

Nëse var është një instancë e Variablat, var.m() do printojë 1 për x dhe 0 për y. Edhe pse variabli y është deklaruar jashtë metodës m(), ai është i aksesueshëm brenda kësaj metode.

Këshillë: Për të shmangur ngatërresat dhe gabimet, mos përdorni emrat e variablave të instancës ose variablave statikë si emra të variablave lokalë, me përjashtim të parametrave të metodave.