JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Luento 10

ITKP102 Ohjelmointi 1

Antti-Jussi Lakanen

Pari muistutusta

- Harkkatyön suunnitelma tulee hyväksyttää ohjaajalla tällä viikolla
- Debuggausnäyte

Taulukot

- Tavallisesti taulukon alkiot liittyvät jollain tavalla loogisesti toisiinsa
- Tämä ja muutama muu taulukko-esimerkki Riderissa

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

C#:n tyyppi-järjestelmästä

Luento 10

- C# tyyppijärjestelmästä
  - Arvopohjaiset tyypit (engl. value types)
  - Viitetyypit (engl. reference types)
- C# ja muistinhallinta
- Pino, keko

C# tyyppijärjestelmästä

- C#:ssa jokaisella muuttujalla on tyyppi
- Tyyppejä ovat esimerkiksi int, double, bool, string ja PhysicsObject
- Tyyppi on joko
  - arvopohjainen tai
  - viitepohjainen

Arvo- ja viitepohjaiset muuttujat

- Arvopohjaisia muuttujia ovat sellaiset muuttujat, joiden tyyppi on (mm.)
  - Int, bool, double, char, …
- Vastaavasti viitepohjaisia muuttujia ovat string, taulukot (esimerkiksi int[]) ja kaikki class-tyyppiset oliot (esimerkiksi PhysicsObject).

C# ja muistinhallinta

- Ohjelman aikana luodut muuttujat vievät tilaa tietokoneen keskusmuistista (RAM)
- .NET ympäristö[1] järjestelee ohjelmassa määriteltyjä muuttujia kahteen sijaintiin:
  - Pinoon
  - Kekoon

1: Tarkemmin sanottuna kyseessä on .NET:n ajonaikainen ympäristö, nimeltään Common Language Runtime (CLR). Tätä voisi kutsua järjestelmäksi, jonka sisällä C#-kielestä käännetty tietokoneohjelma operoi käyttöjärjestelmän päällä.

Pino ja keko

- Pino (engl. *stack*) ja keko (engl. *heap*) ovat .NET-alustan ja C#-kielen muistinhallintaan liittyviä tietorakenteita
- Vaikuttavat siihen, miten ohjelmat tallentavat ja käsittelevät tietoa

Pino (stack)

- Pino on alue muistissa, jota käytetään paikallisten muuttujien tallentamiseen sekä metodikutsujen hallintaan. Se toimii LIFO-periaatteella (Last In, First Out), mikä tarkoittaa, että viimeksi lisätty elementti poistetaan ensimmäisenä.
- Aliohjelman paikalliset muuttujat ja paluuarvon osoitin tallennetaan pinoon. Nämä tiedot poistuvat aliohjelman suorituksen päättymisen jälkeen
- Pino on nopea, koska sen hallinta on yksinkertaista ja ennustettavaa, mutta sen koko on rajattu ja se voi loppua kesken (stack overflow), jos rekursiota tai paikallisten muuttujien määrää ei hallita.

Keko (heap)

- Keko on muistin alue, jota käytetään dynaamisesti allokoidun muistin, kuten olioiden ja taulukoiden, tallentamiseen. Se ei noudata LIFO-periaatetta kuten pino, ja muistiin päästään käsiksi suoraan osoitteiden avulla.
- Kun C#-ohjelmassa luodaan uusi olio new-avainsanalla, se tallennetaan kekoon. Keko mahdollistaa muistin dynaamisen allokoinnin ja vapauttamisen, mikä tarkoittaa, että ohjelma voi pyytää ja vapauttaa muistia tarpeen mukaan.
- Kekomuistin hallinta on hitaampaa kuin pinomuistin, koska se vaatii muistin allokointia ja vapautusta sekä mahdollista muistin roskien keräystä (garbage collection) välttämättömän muistin vapauttamiseksi.

Pino ja keko

- Väljästi ja aavistuksen epätarkasti voisi kuvailla seuraavasti:
- Pino on tietorakenne, jonka vastuulla on pitää yllä tietoa ”juuri tämän hetkisestä” tilanteesta: mitä funktiota on kutsuttu, mistä on kutsuttu, millä parametreilla, käytettävissä olevat muuttujat, jne.
- Keko on tietorakenne, jonka vastuulla on säilyttää ”dataa”, siis ohjelman aikana syntynyttä ja käytettävää muuttujien sisältämää tietoa.

Pino- ja kekorakenteiden merkitys käytännön ohjelmoinnissa

- Arvopohjaisten muuttujien arvot tallennetaan pinoon.
- Viitepohjaisten muuttujien arvot tallennetaan kekoon, ja viite arvoon tallennetaan pinoon.
- Keko on välttämätön suurten tietomäärien käsittelyssä
- Keko mahdollistaa ohjelman varata muistia tarpeen mukaan ja vapauttaa sen, kun sitä ei enää tarvita. Tämä on erityisen tärkeää suurten tietomäärien käsittelyssä ja oliopohjaisessa ohjelmoinnissa, missä olioiden elinkaari voi vaihdella suuresti.

Arvopohjainen tyyppi

- Sisältää ”suoraan” oman datansa.
- Esimerkiksi jos kirjoitamme int a = 3;
- Tässä a:n arvo on 3.

Viitepohjainen tyyppi (1/3)

- Sisältää viitteen, ts. sisältää ”epäsuorasti” oman datansa.

Viitepohjainen tyyppi (2/3)

- Kaksi muuttujaa (esimerkiksi a ja b) voi viitata samaan dataan.
- Tästä seuraa, että a-muuttujan kautta dataan tehdyt muutokset heijastuvat myös b-muuttujaan.

Viitepohjainen tyyppi (3/3)

- Muun muassa taulukot ovat viitepohjaisia tyyppejä.
- Esimerkiksi jos kirjoitamme int[] a = new int[5];
- Tässä a:n arvo on todellisuudessa viite sisältöön, jossa viiden mittainen taulukko todella sijaitsee.

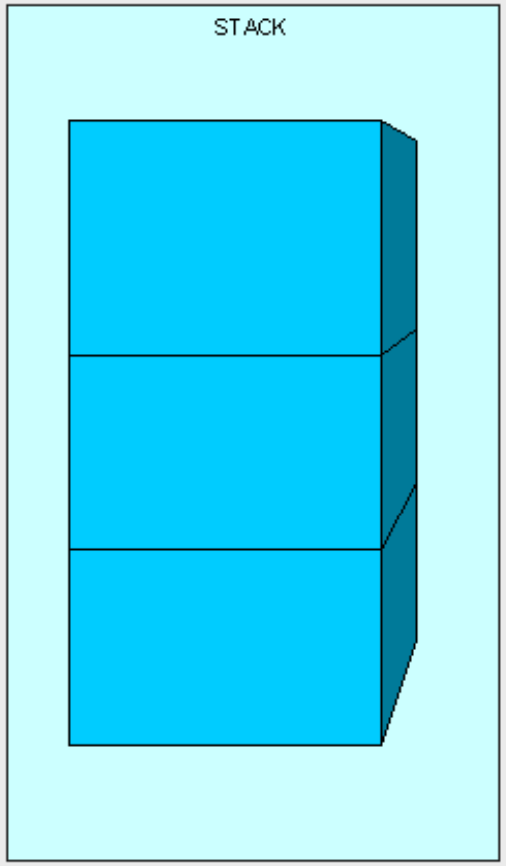
Sijoitus (arvopohjainen)

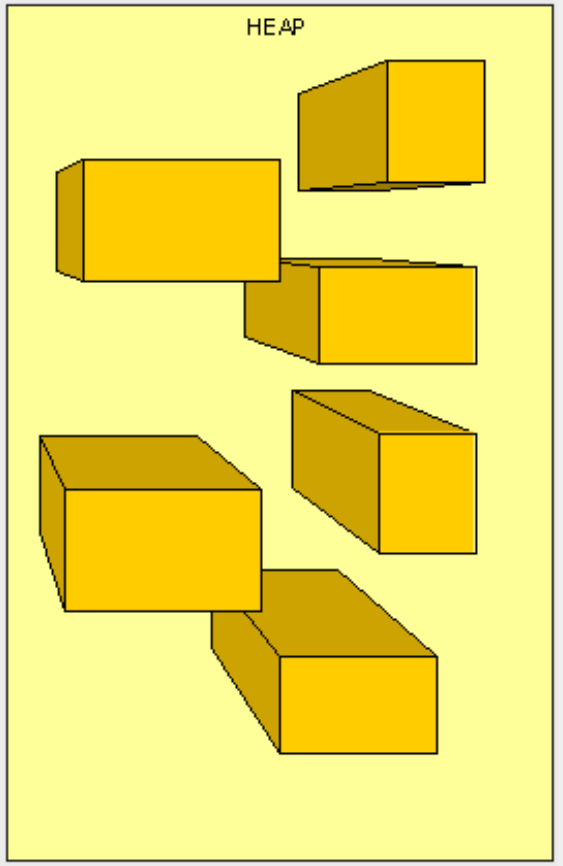
- int a = 3;
- int b = a; // a:n arvo (3) kopioidaan
- b = b+1; // a:n arvo säilyy

Sijoitus (viitepohjainen)

- int[] taulukko1 = { 1, 2, 3 };
- int[] taulukko2 = taulukko1;
- taulukko2[1] = 4;
- // Mitä nyt on taulukko1[1]???

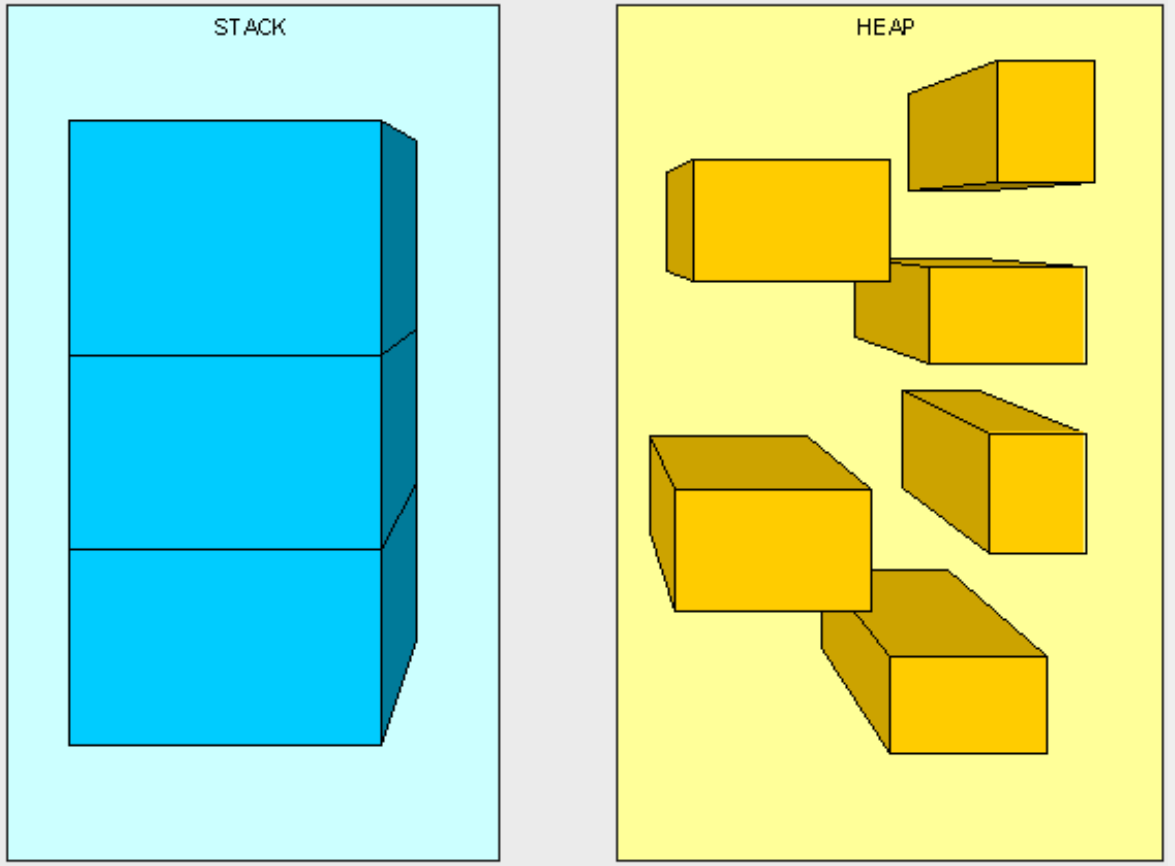
Koodi





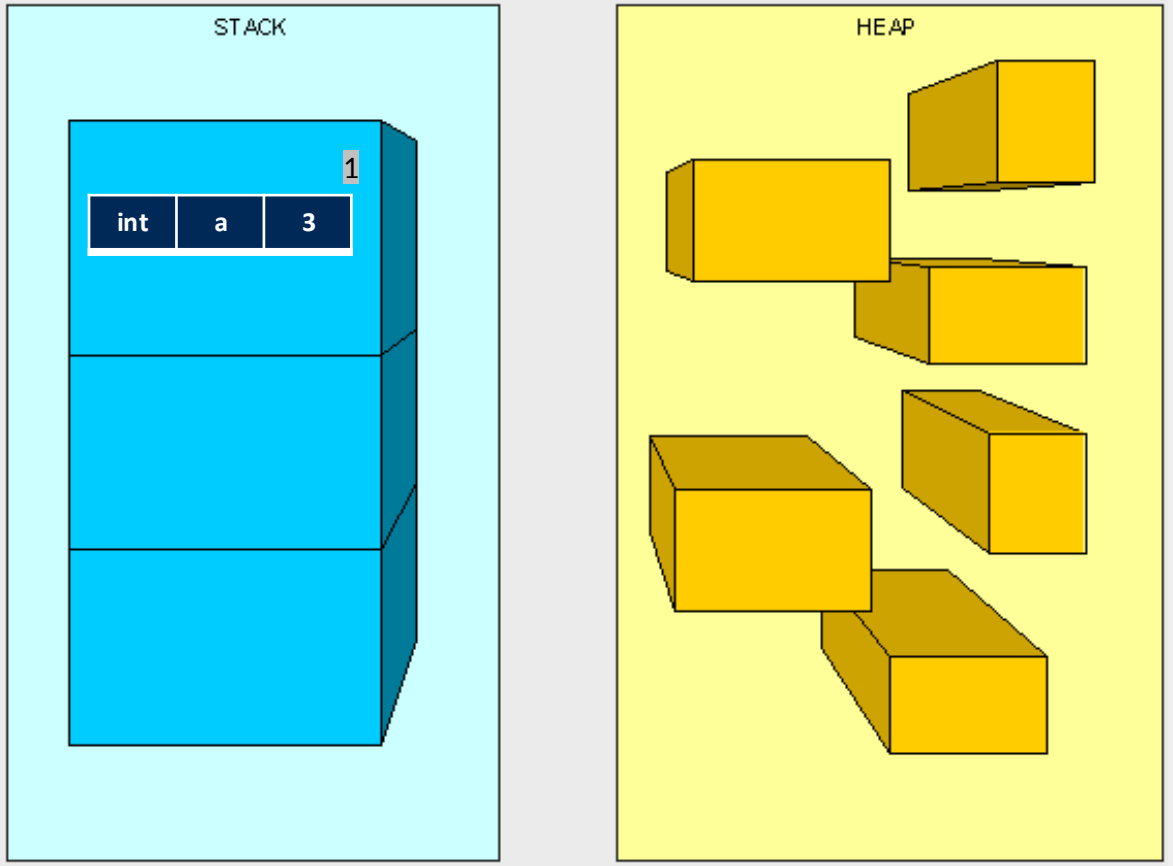
Koodi

```
int a = 3;
```



Koodi

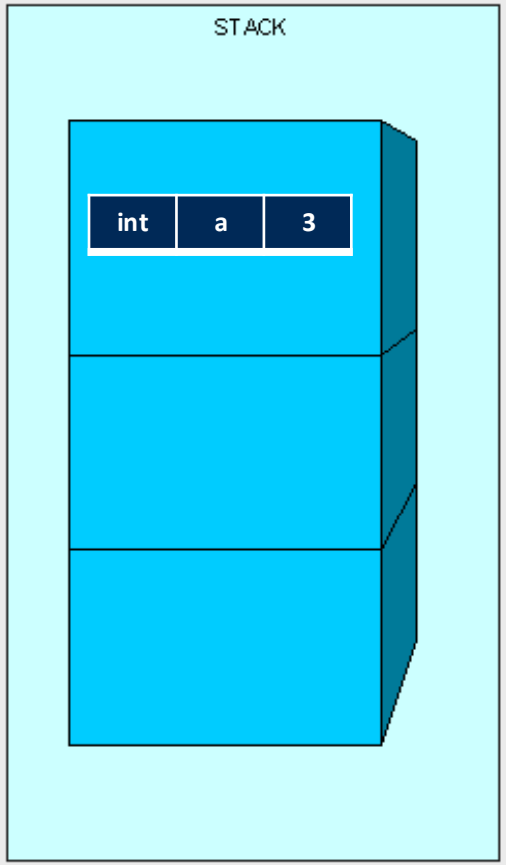
```
int a = 3;
```

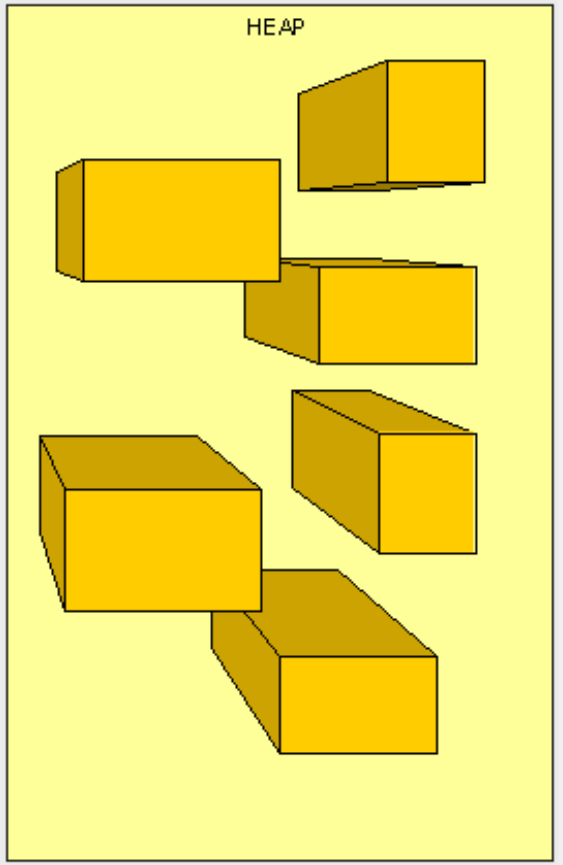


1: Tarkasti ottaen arvopohjaisten muuttujien arvot voivat sijaita joko pinossa tai keossa riippuen siitä, missä kontekstissa muuttuja on määritelty. Esimerkiksi Henkilö-luokka (viitepohjainen, sijaitsee keossa) voisi sisältää int-tyyppisen ikä-attribuutin. Tässä tilanteessa myös ikä sijaitsisi keossa, ei pinossa. Tämän esimerkin tarkoitus on kuitenkin havainnollistaa

Koodi

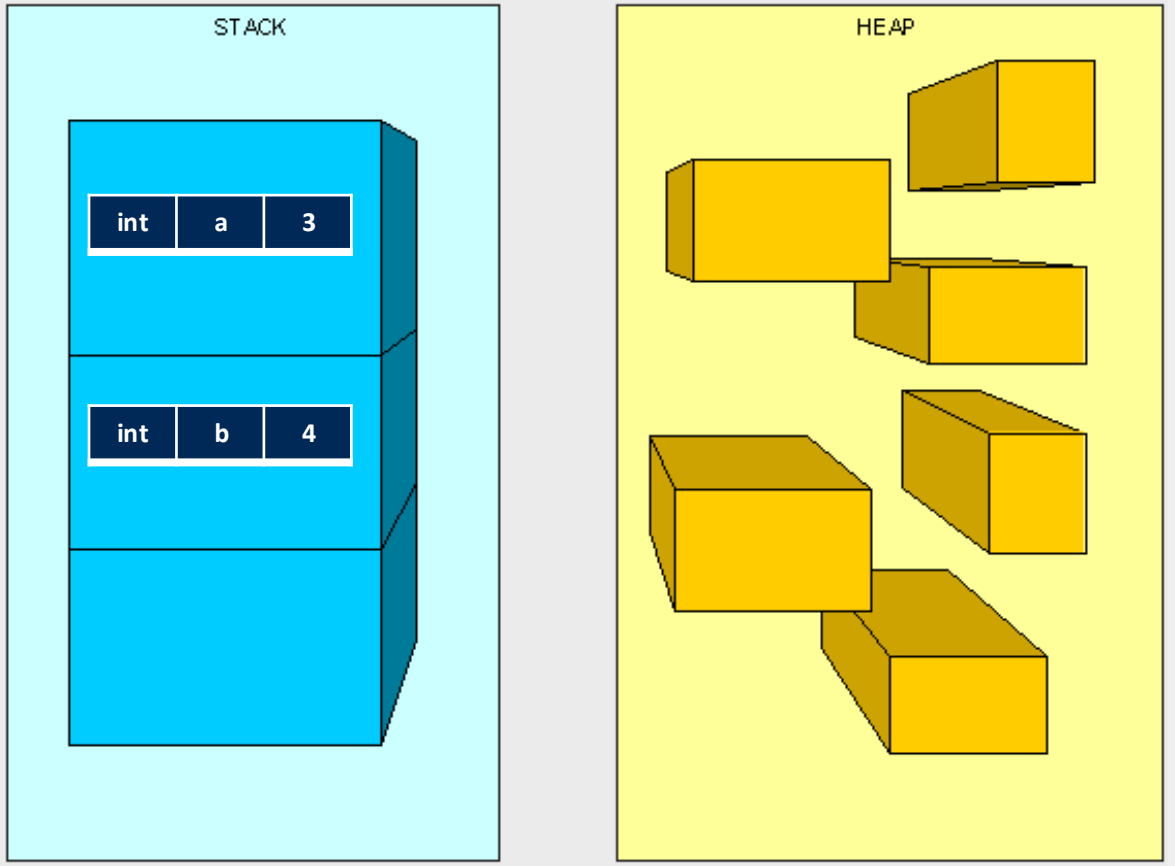
```
int a = 3;
int b = 4;
```





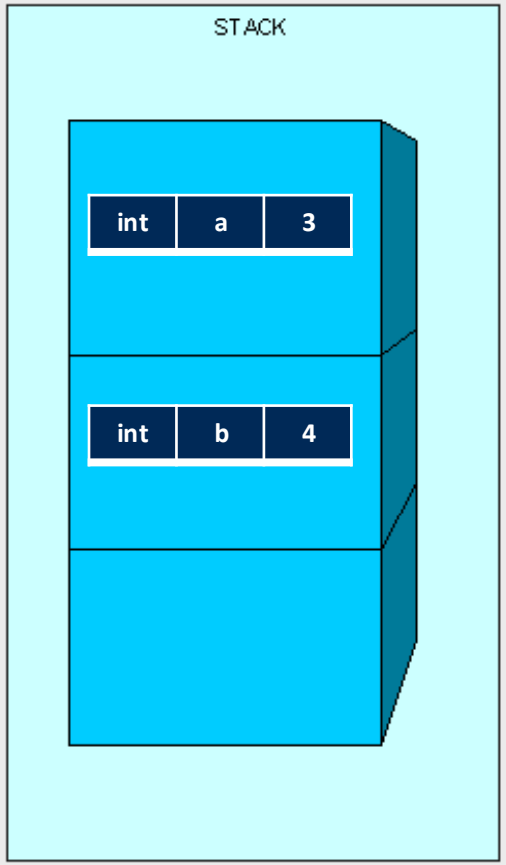
Koodi

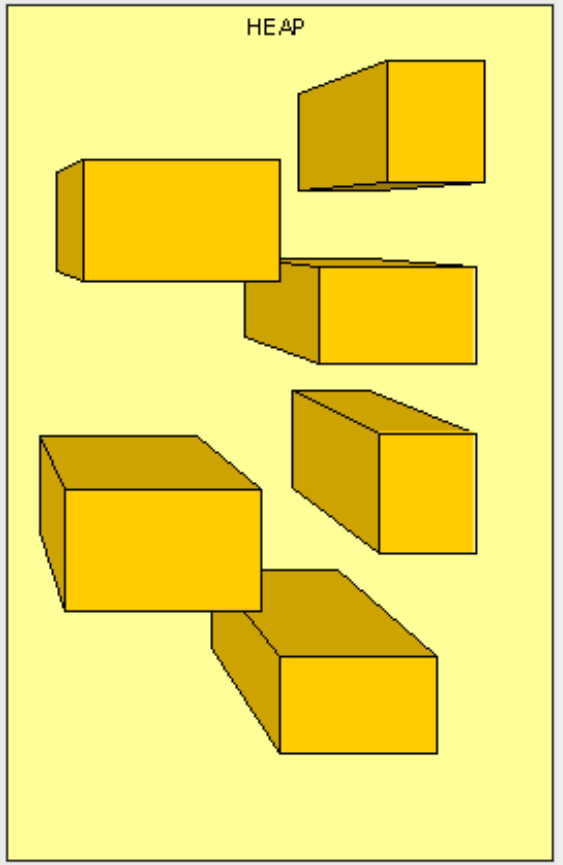
```
int a = 3;
int b = 4;
```



Koodi

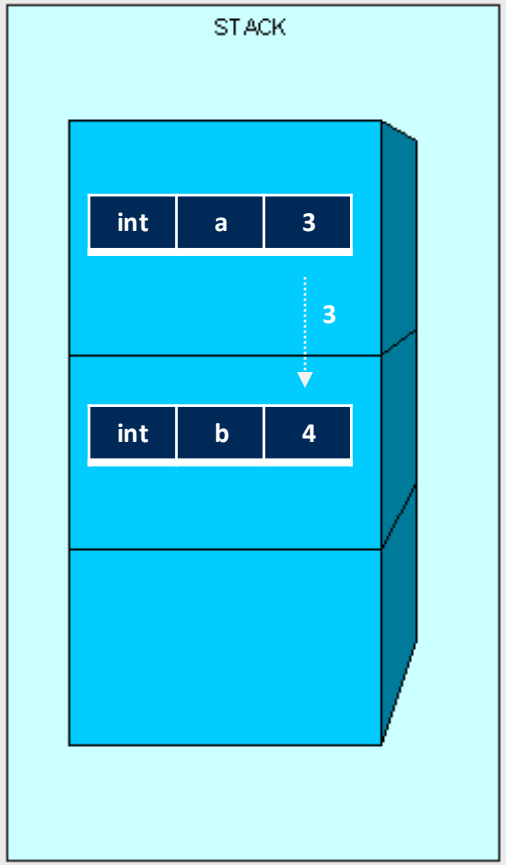
```
int a = 3;
int b = 4;
b = a;
```

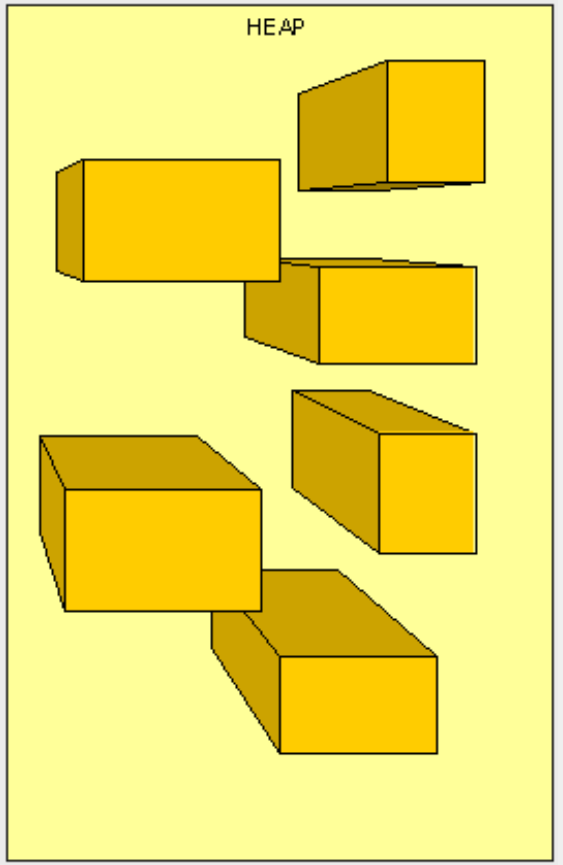




Koodi

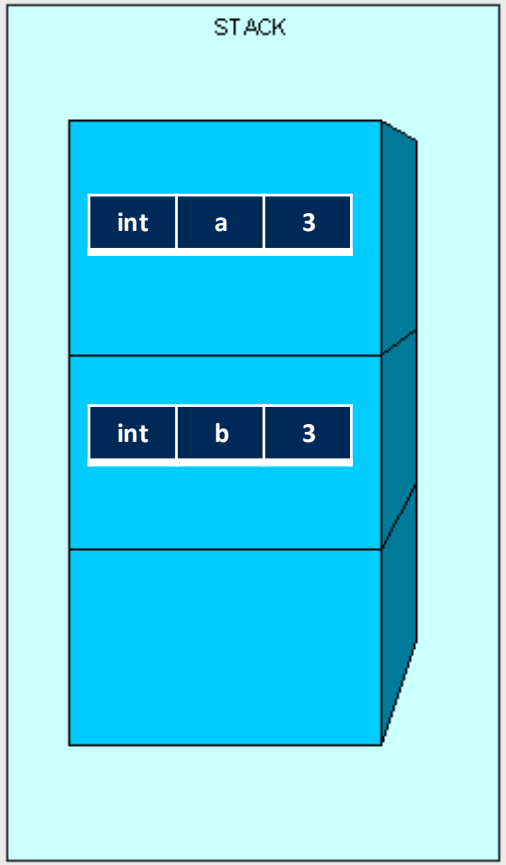
```
int a = 3;
int b = 4;
b = a;
```

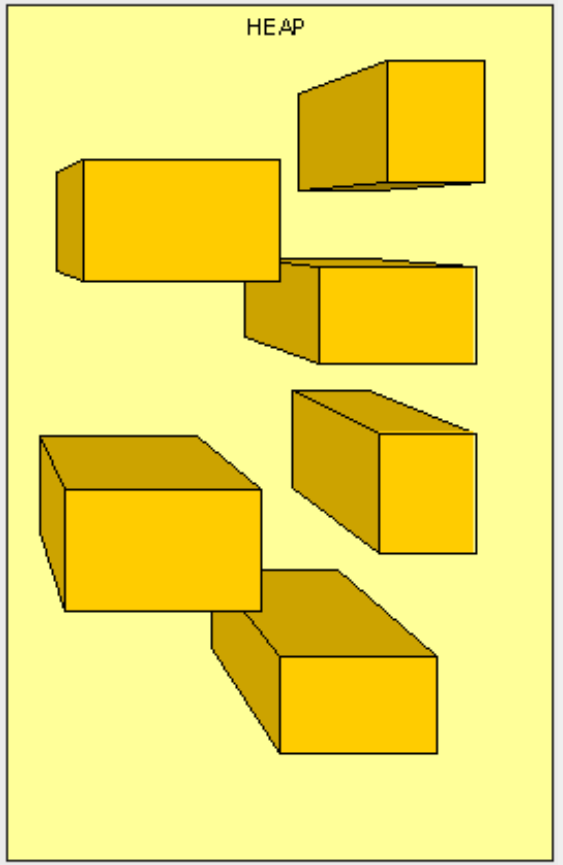




Koodi

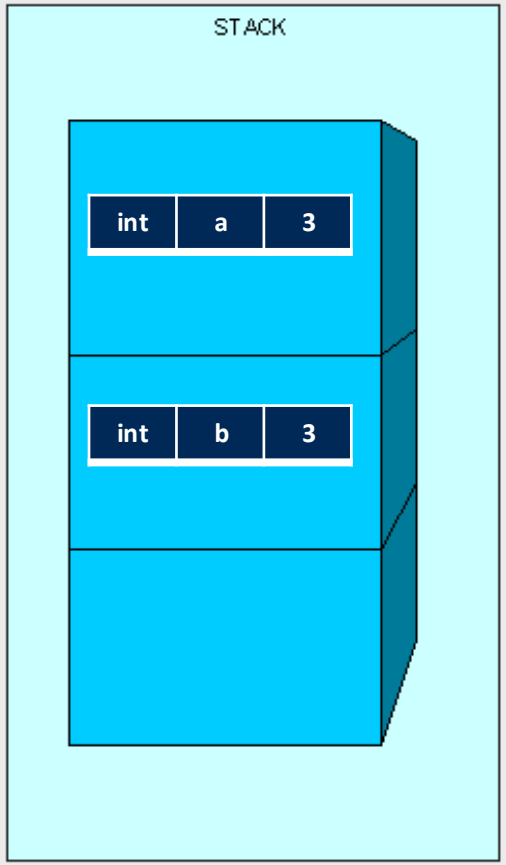
```
int a = 3;
int b = 4;
b = a;
```

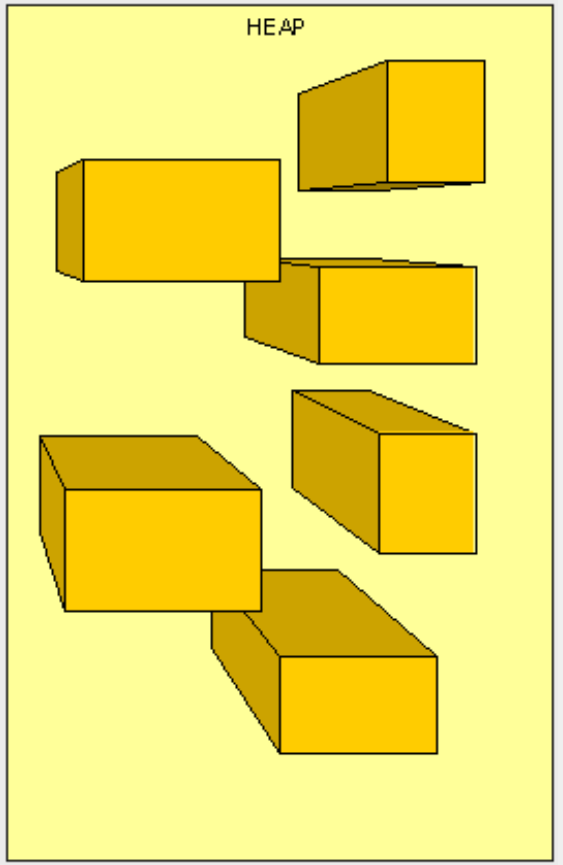




Koodi

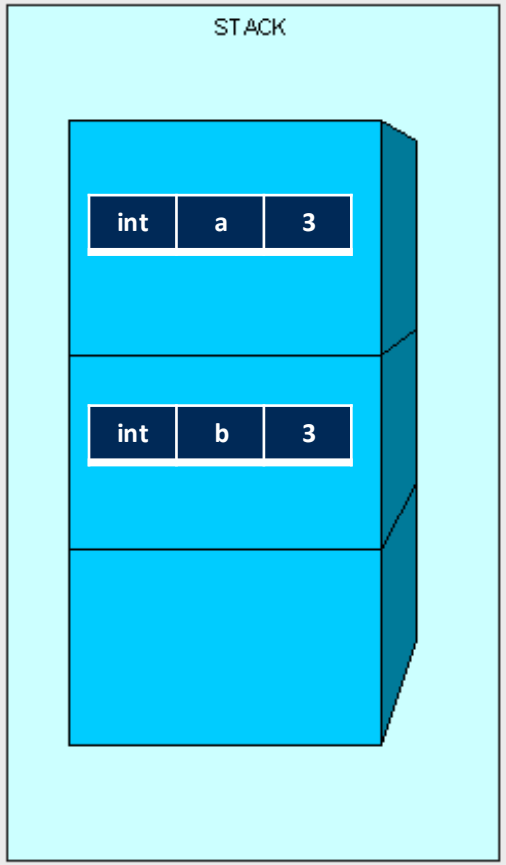
```
int a = 3;
int b = 4;
b = a;

int[] t = new int[]
{1, 2, 3};
```

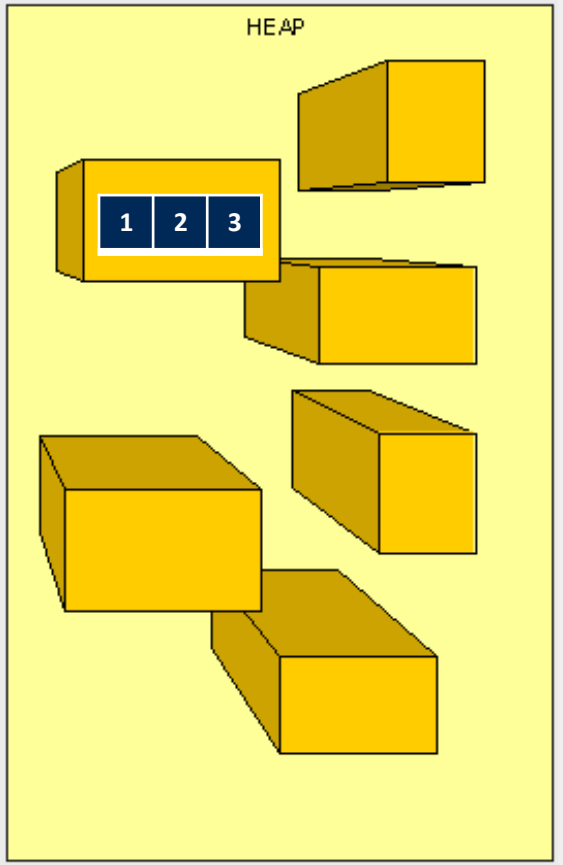




Koodi

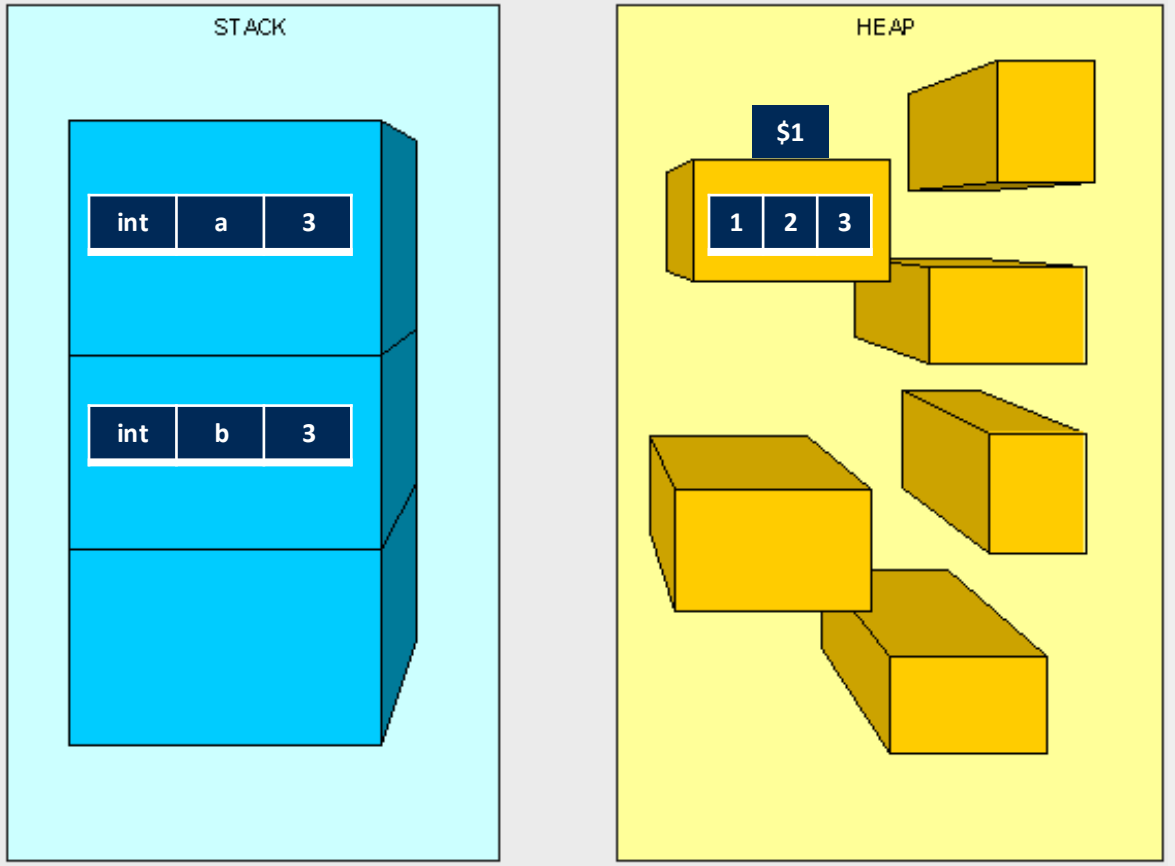
```
int a = 3;
int b = 4;
b = a;

int[] t = new int[]
{1, 2, 3};
```





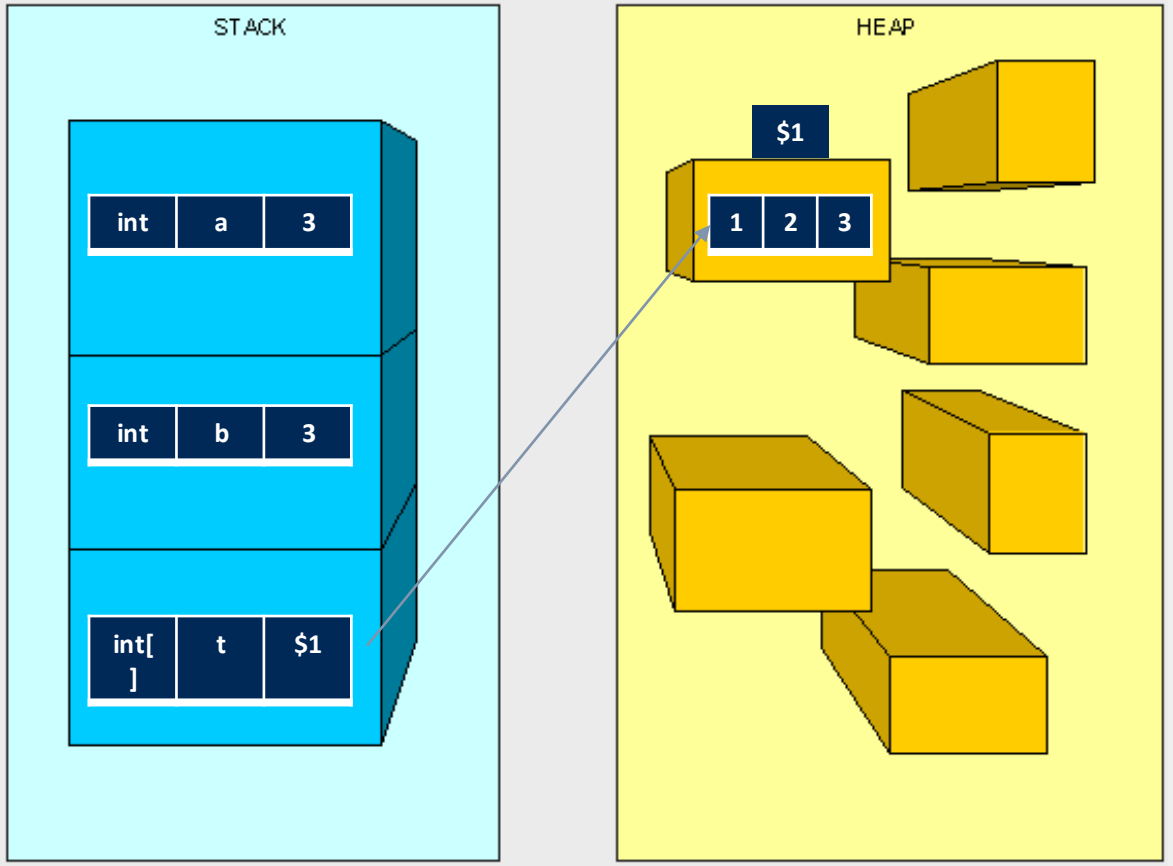
Koodi

```
int a = 3;
int b = 4;
b = a;

int[] t = new int[]
{1, 2, 3};
```



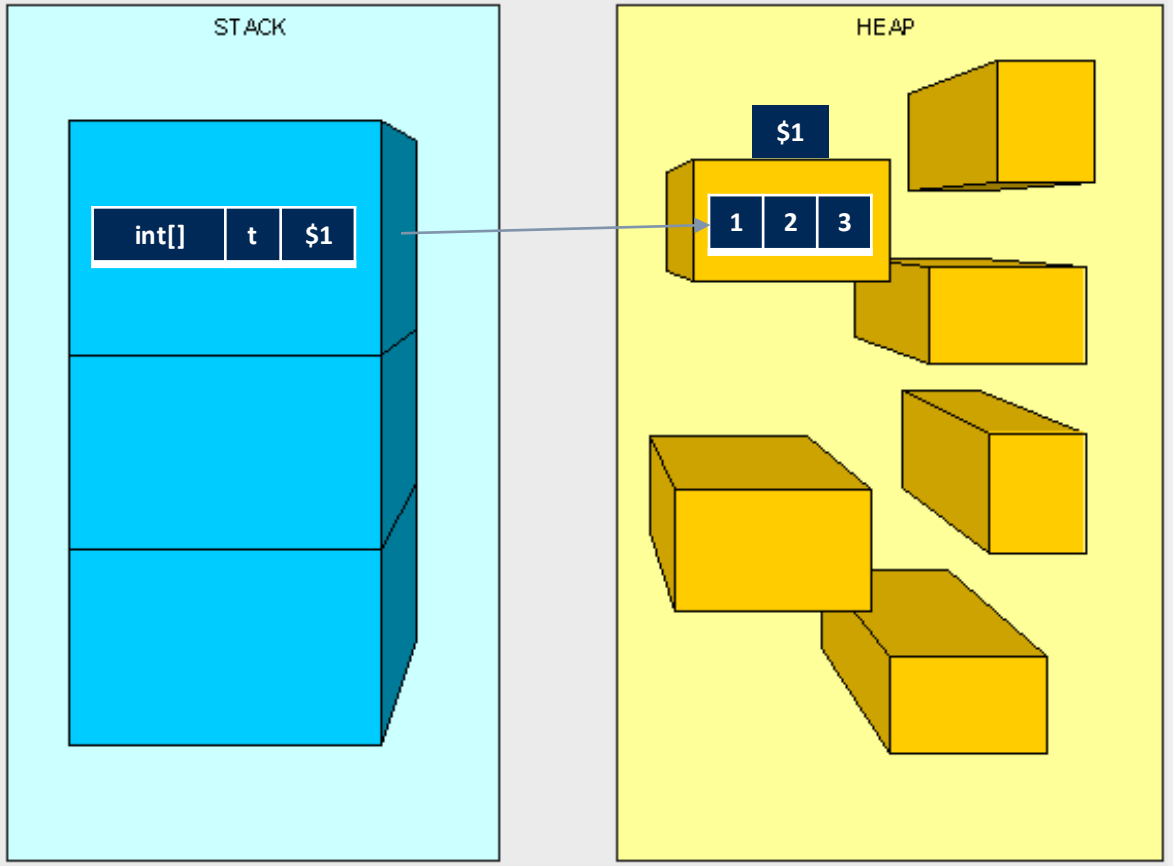
Koodi

```
int a = 3;
int b = 4;
b = a;

int[] t = new int[]
{1, 2, 3};
```



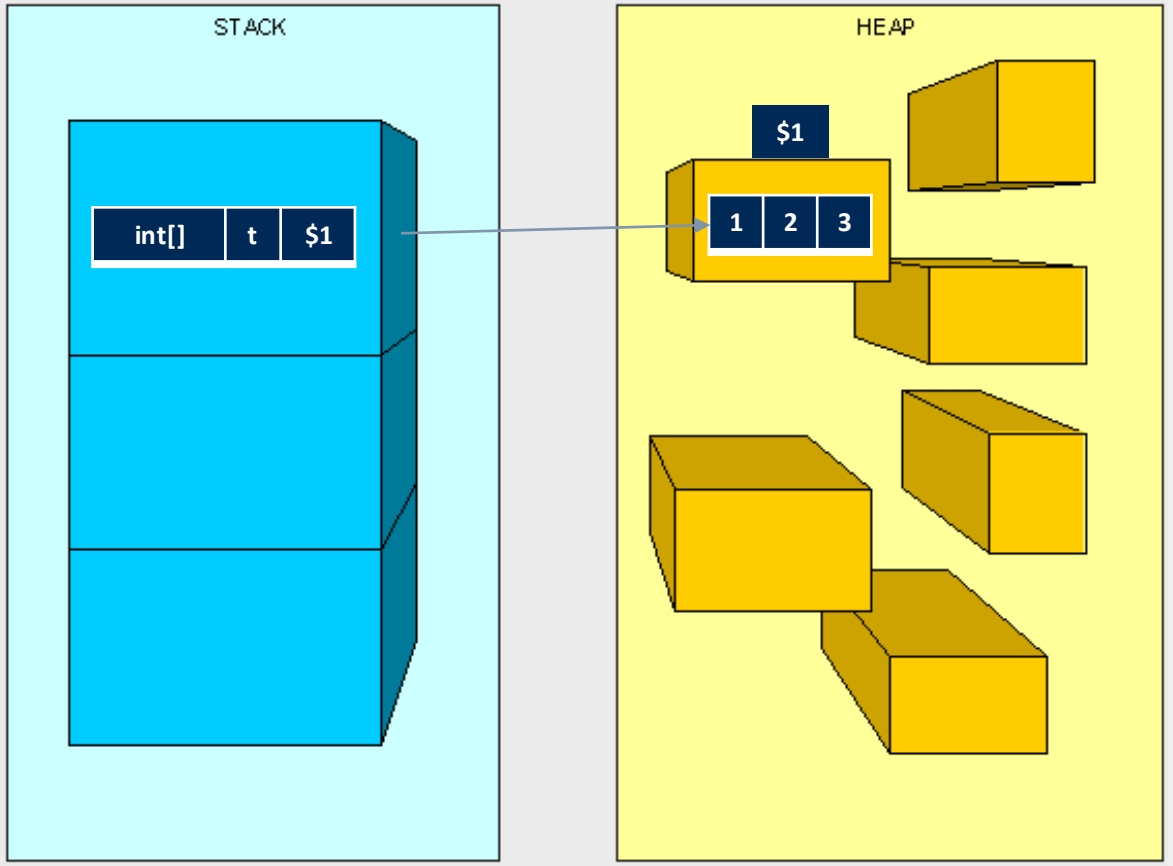
Koodi

```
int[] t = new int[]
{1, 2, 3};
```



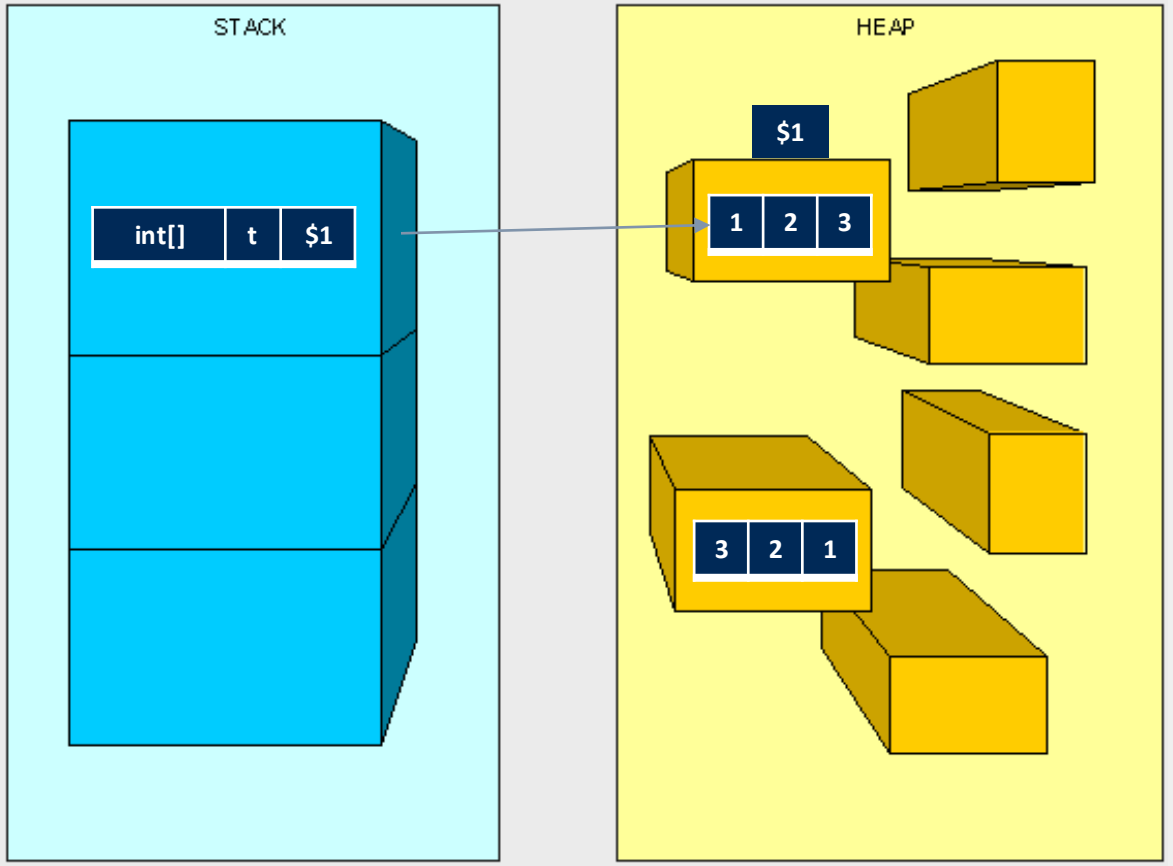
Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};
```



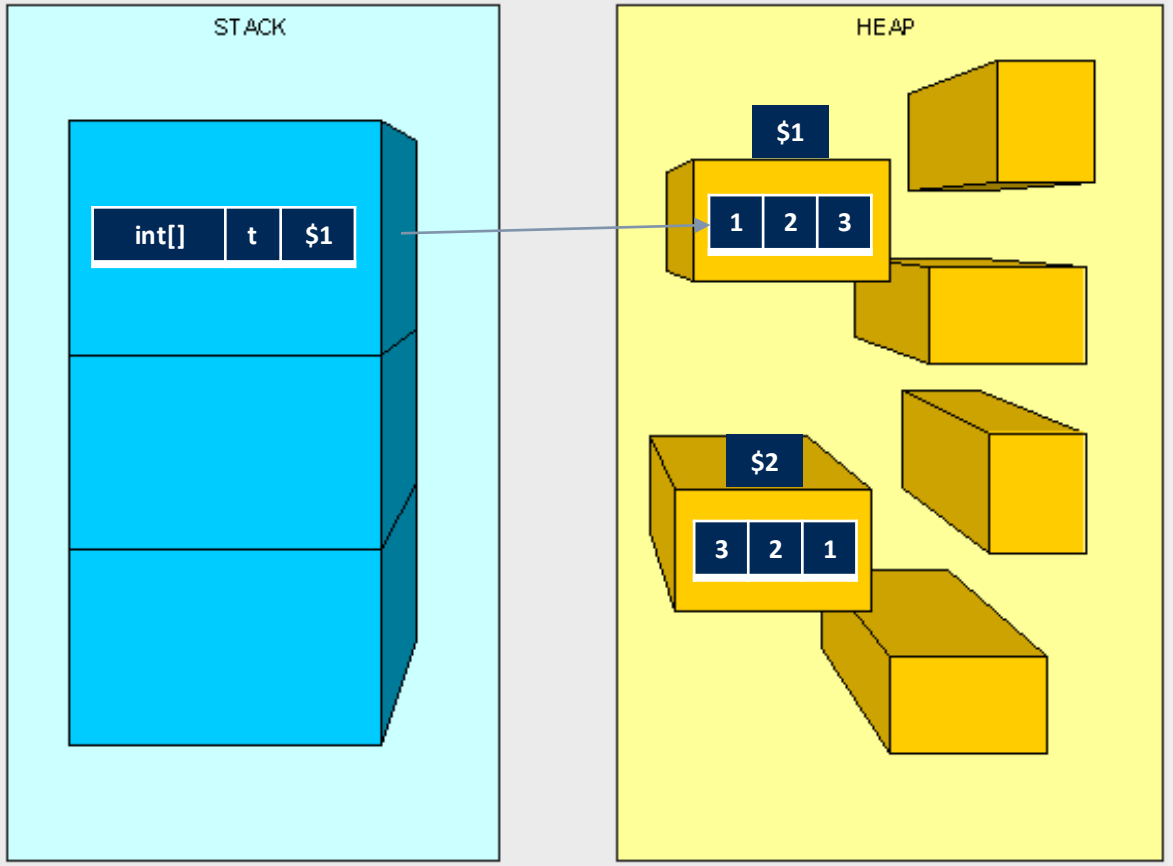
Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};
```



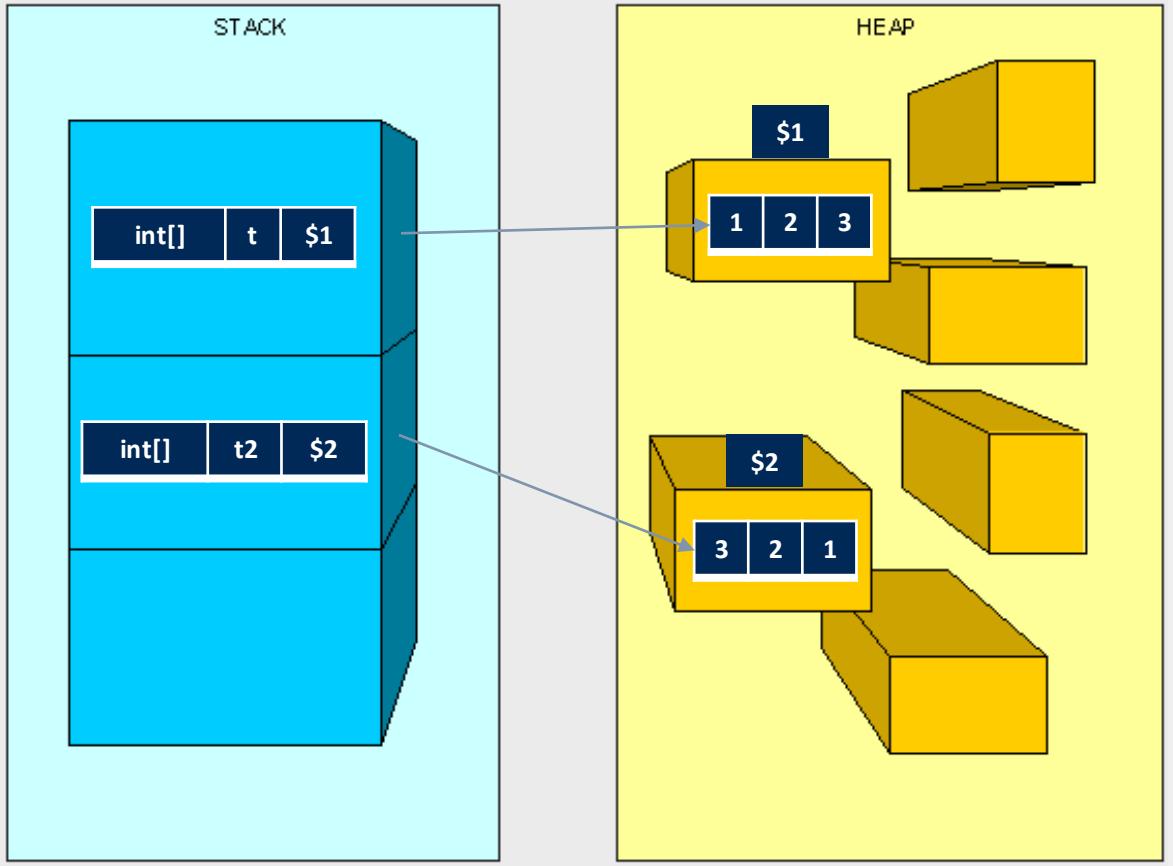
Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};
```



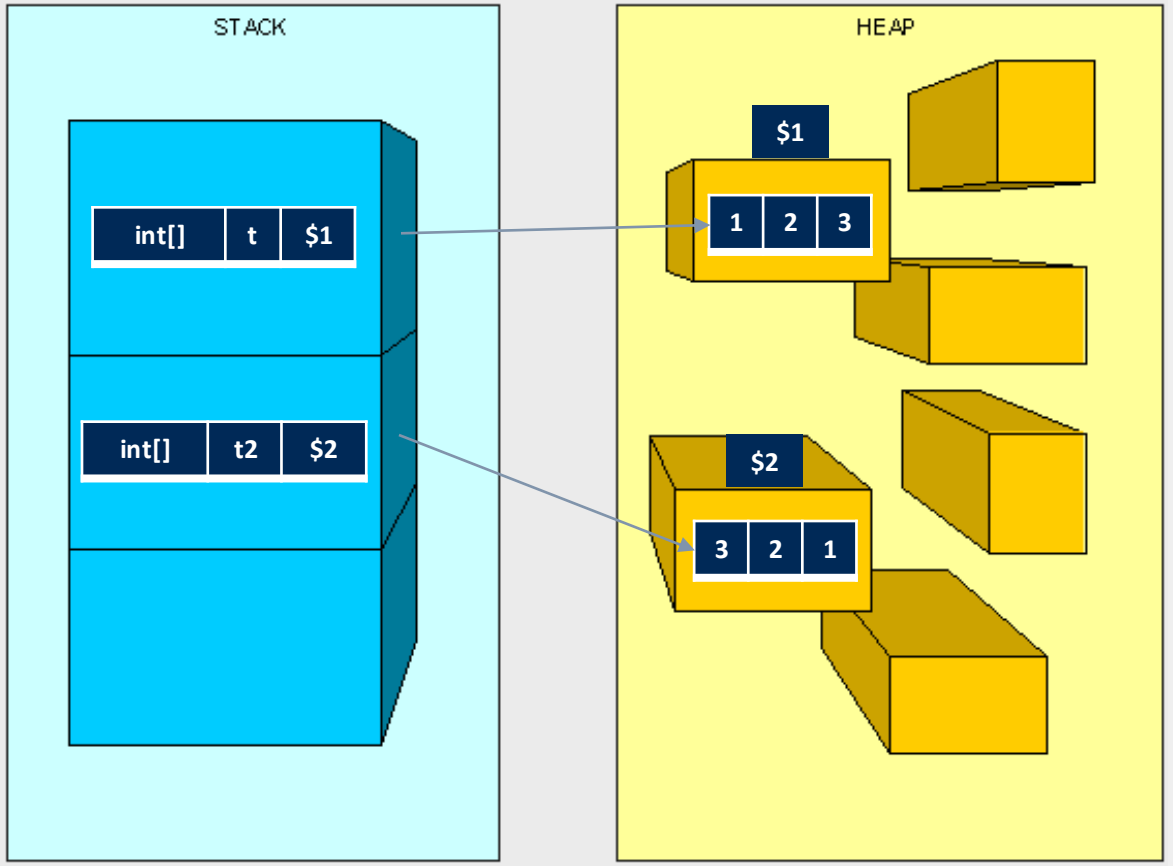
Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};
```



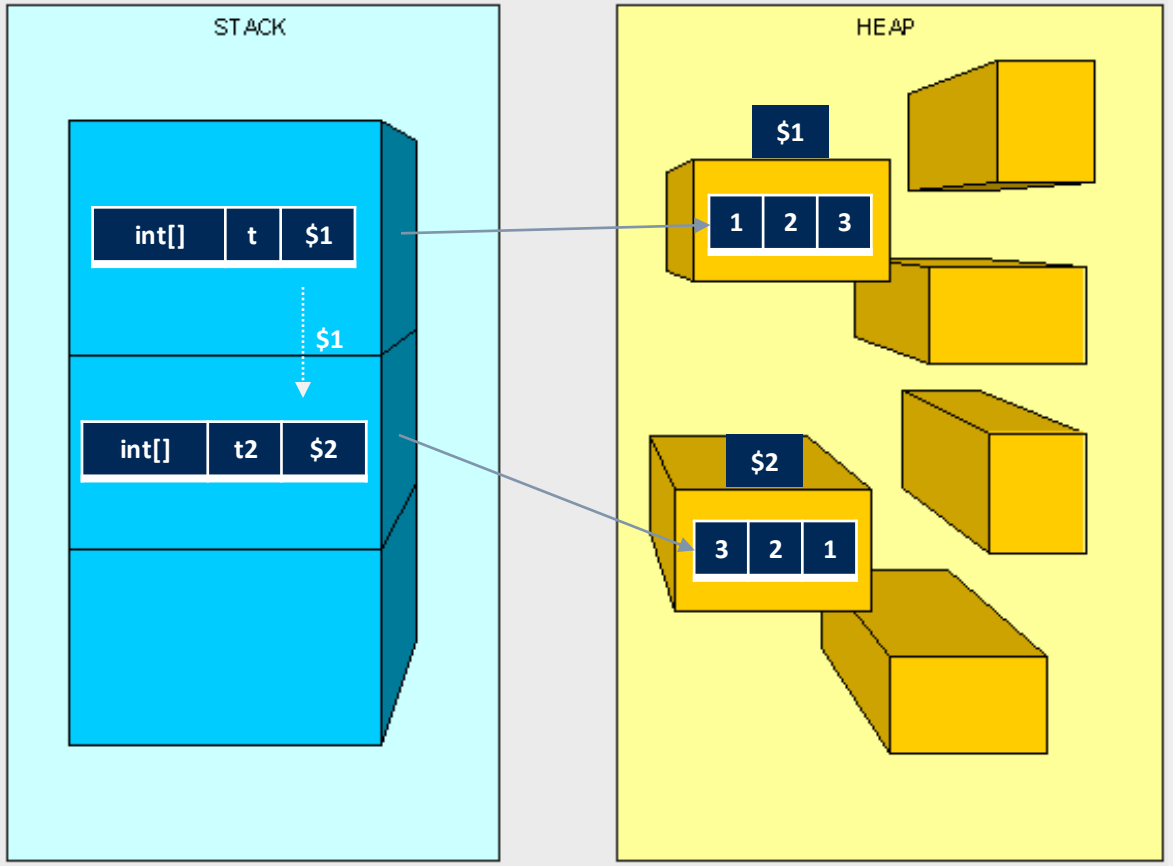
Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};

t2 = t;
```



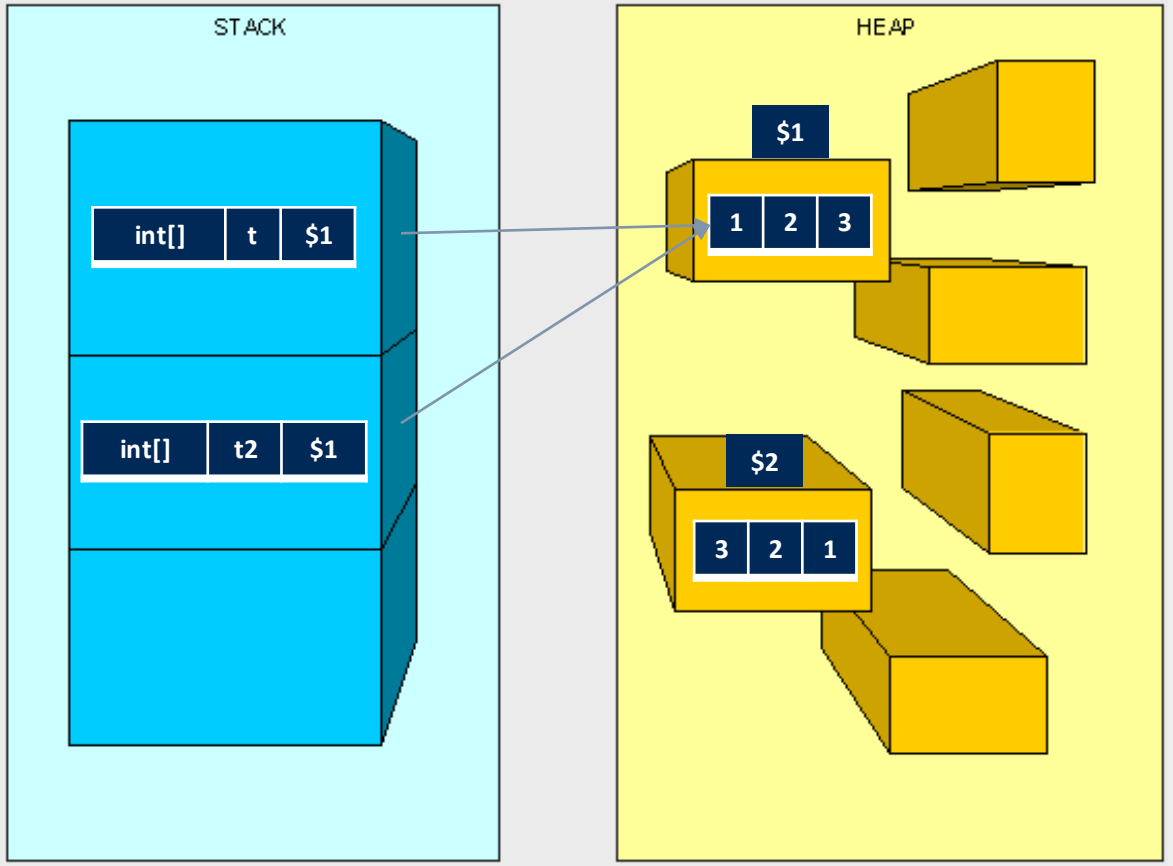
Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};

t2 = t;
```



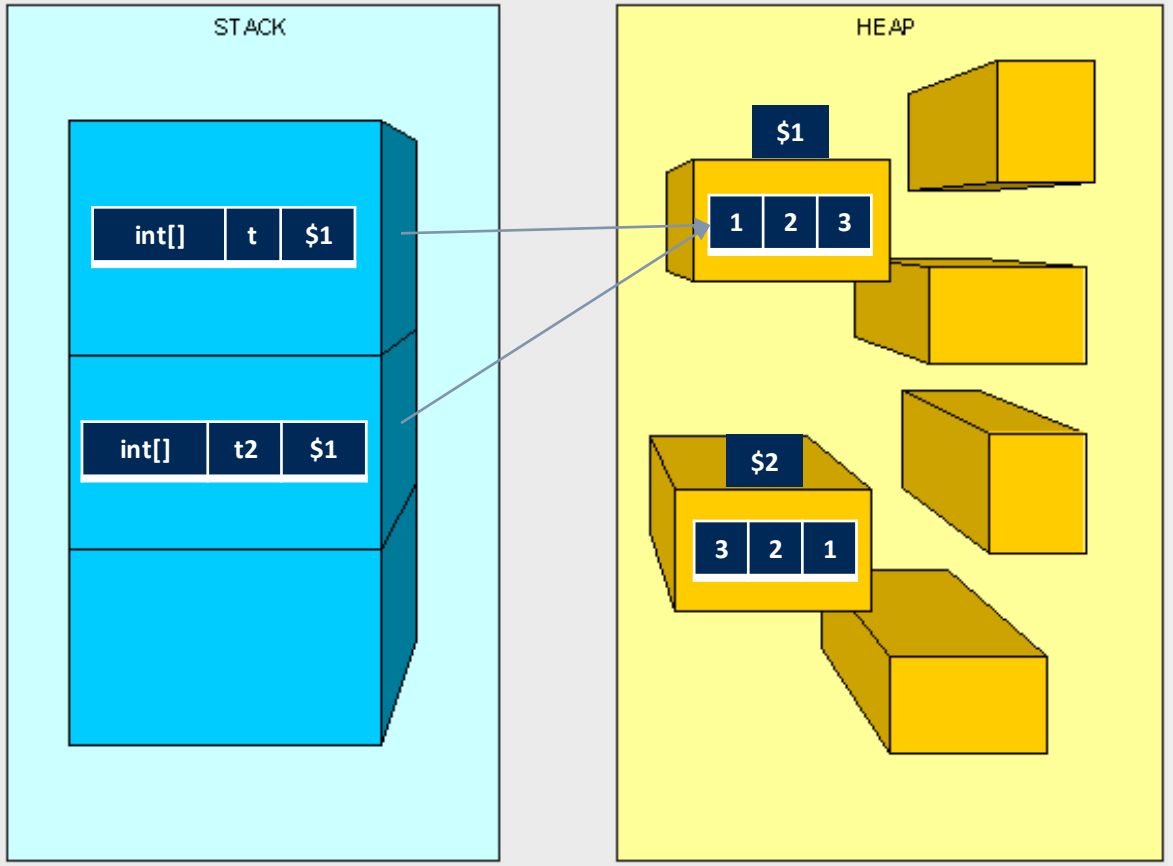
Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};

t2 = t;
```



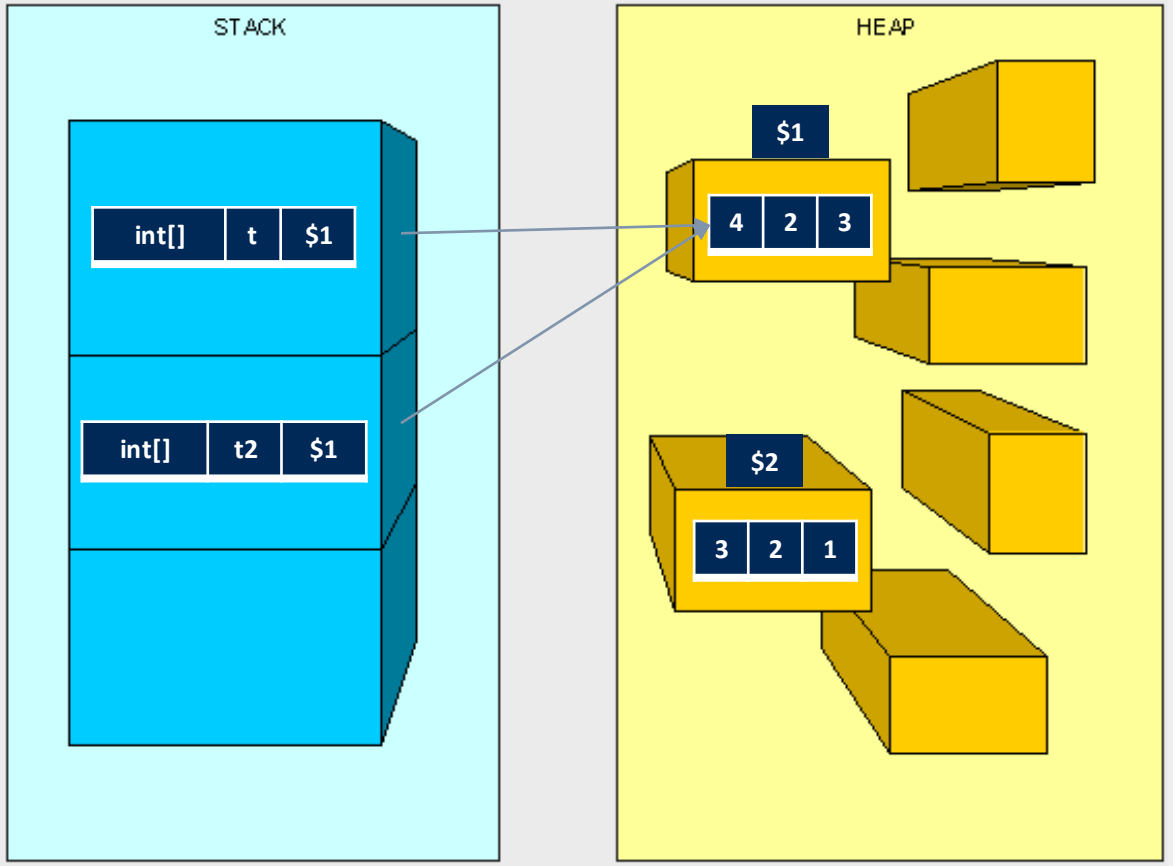
Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};

t2 = t;

t2[0] = 4;
```



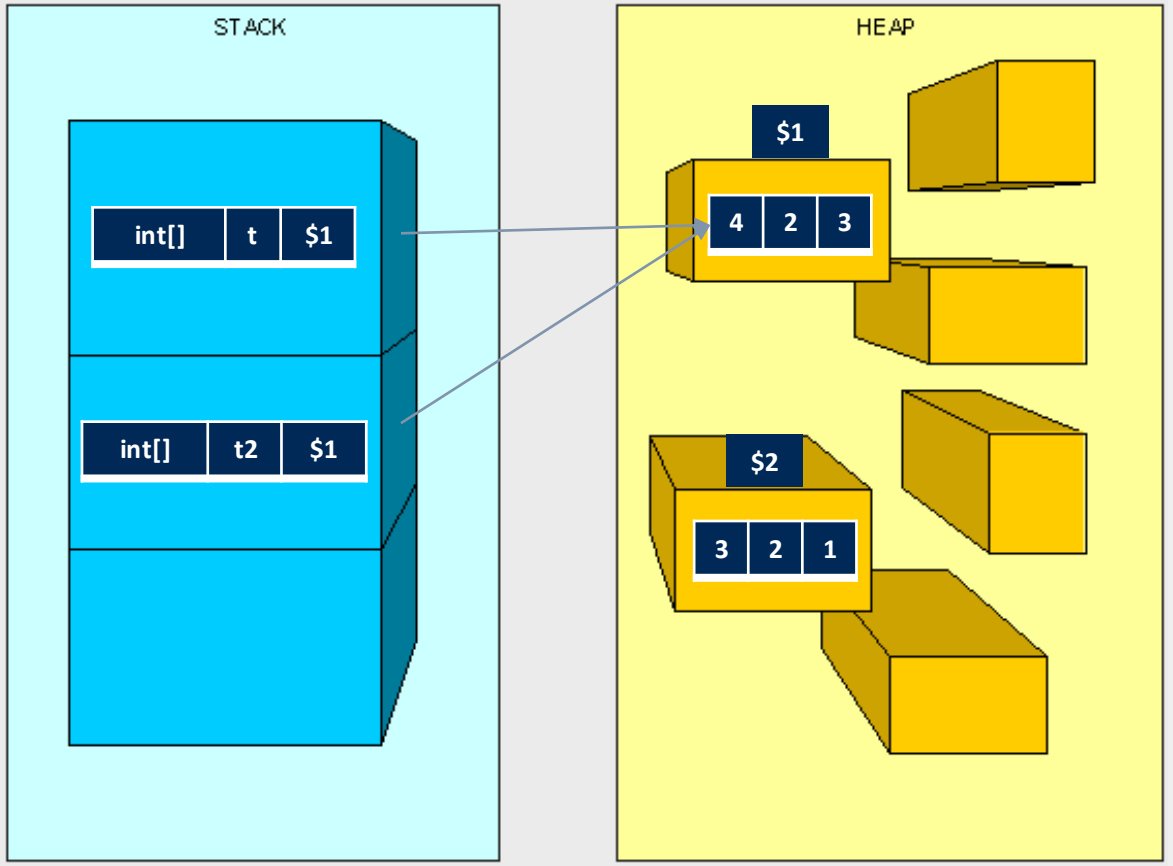
Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};

t2 = t;

t2[0] = 4;
```



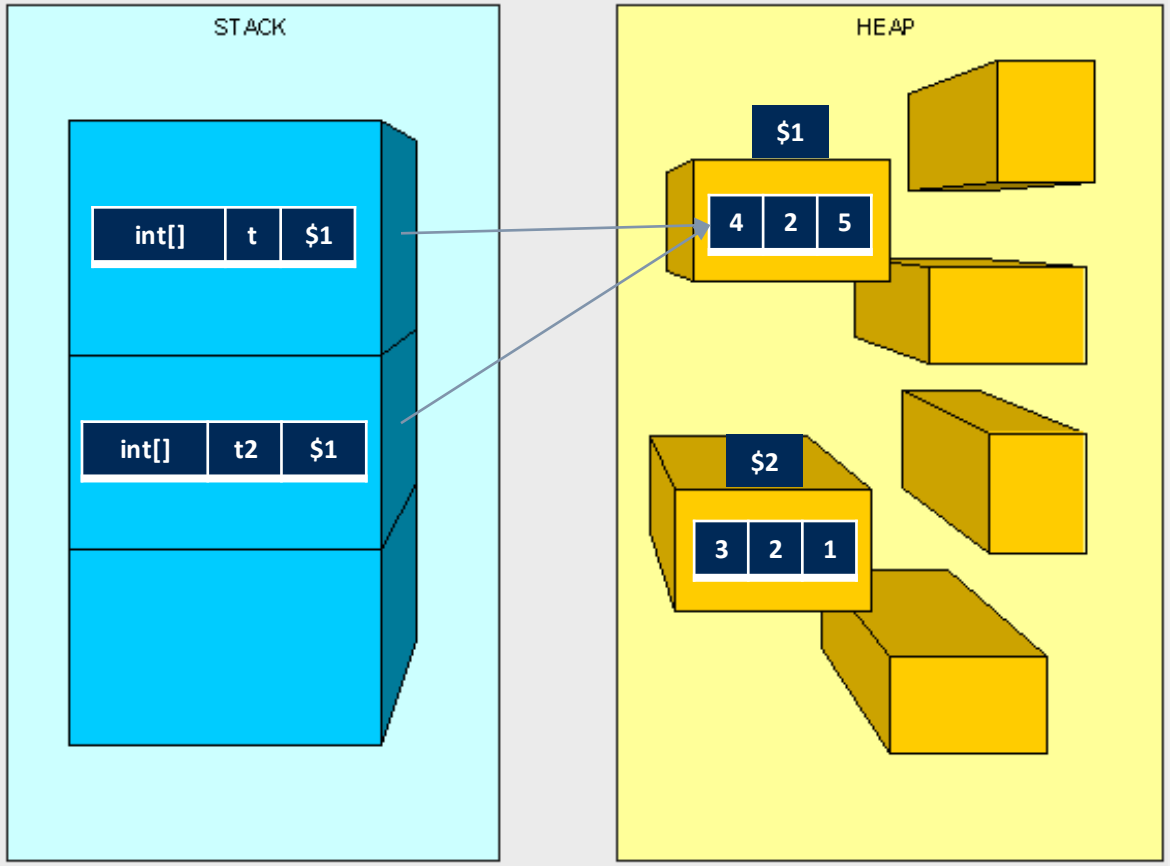
Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};

t2 = t;

t2[0] = 4;

t[2] = 5;
```



Koodi

```
int[] t = new int[]
{1, 2, 3};

int[] t2 = new
int[] {3, 2, 1};

t2 = t;

t2[0] = 4;

t[2] = 5;
```



Automaattinen roskienhallinta

- Muisti, johon ei enää ohjelmassa viitata, merkitään automaattisesti ”roskaksi”.
- Edellisessä esimerkissä viitteen $2 vaatima alue muuttui roskaksi.
- C# järjestää automaattisesti muistia sillä tavoin, että uusia muuttujia voidaan taas allokoida kekomuistiin.

KEKO:
---][-------][----------][-----].....
obj 1 obj 2 obj 3 vapaa

KEKO:
---][-------].....[-----].....
obj 1 vapaa obj 3 vapaa

KEKO:
---][-------].....[-----][------------------]...
obj 1 vapaa obj 3 obj 4

KEKO:
---][-------][-----][------------------].....
obj 1 obj 3 obj 4 vapaa

Automaattinen roskienhallinta

- Ohjelmoijan ei aktiivisesti tarvitse pohtia muistinkulutusta
- Tämä ei johdu niinkään nykykoneiden suurista keskusmuistin (RAM) määristä, vaan automaattisesta roskienkeruusta
- Aikakriittisissä järjestelmissä roskienkeruun vaatima aikaresurssi on kuitenkin otettava huomioon.