

Paměť v objektových jazycích

OMO, LS 2013/2014

Alokace Paměti

- Statická alokace - paměť je přidělena prakticky v “době kompilace”
 - Nelze použít pro dynamické datové struktury
- Alokace na zásobníku - dynamická alokace na zásobníku v rámci současného rámce
 - Po skončení volání je paměť dealokována
- Alokace na haldě - dynamická alokace na haldě, alokovanou paměť je třeba spravovat
 - V Java to za nás dělá Garbage Collector

Paměťový model

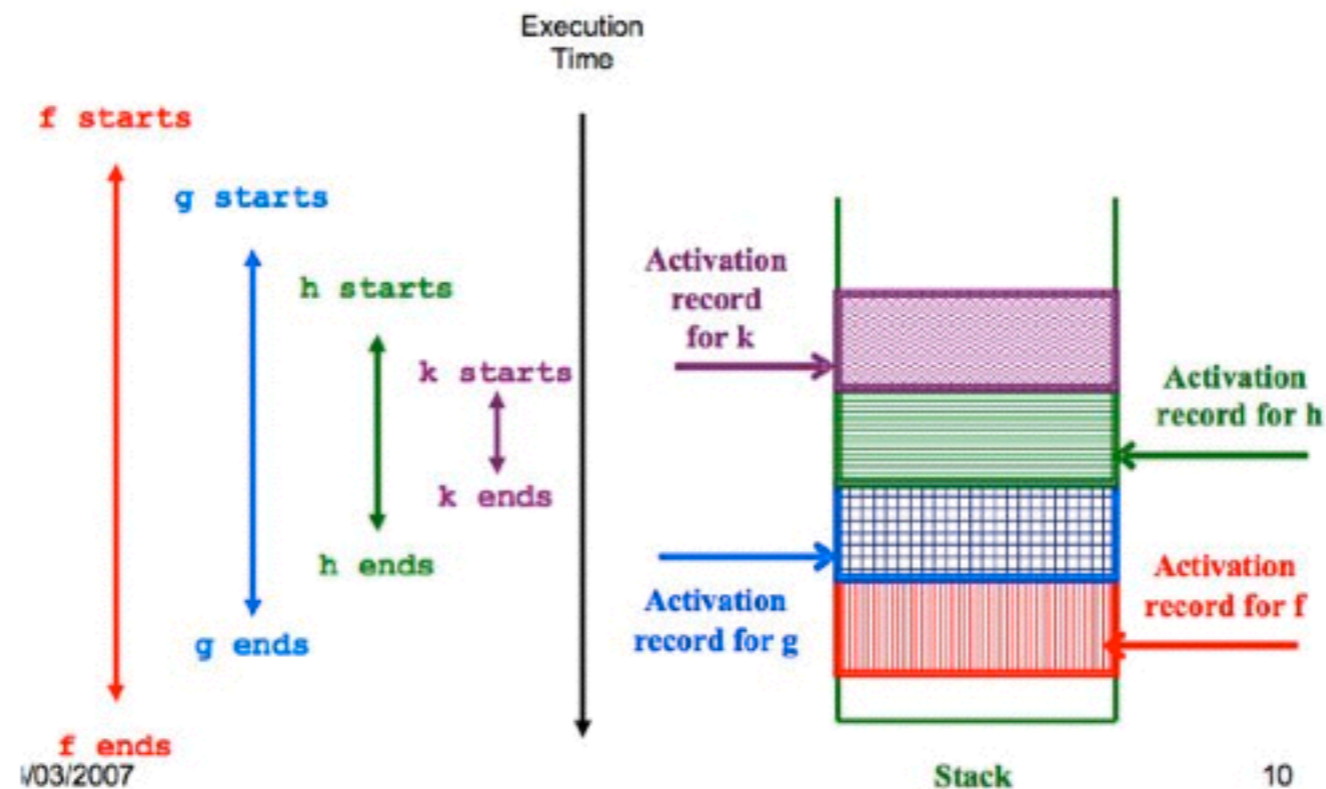
- Paměť rozdělena do tří oblastí (address space)
 - Code area (Prováděný kód)
 - Stack (Zásobník)
 - Heap (Halda)
- Tento model je zjednodušený!
- Budeme uvažovat programy běžící v jednom vlákně.

Activation Record

- Když je metoda volána, tak veškeré informace nutné pro její běh jsou umístěny na zásobník.
- Tyto informace se nazývají Activation Record (AR).
- Když volání skončí (return), pak je AR odstraněn ze zásobníku.

Activation Record

```
void f(){  
    g();  
}  
  
void g(){  
    h();  
}  
  
void h(){  
    k();  
}
```



V03/2007

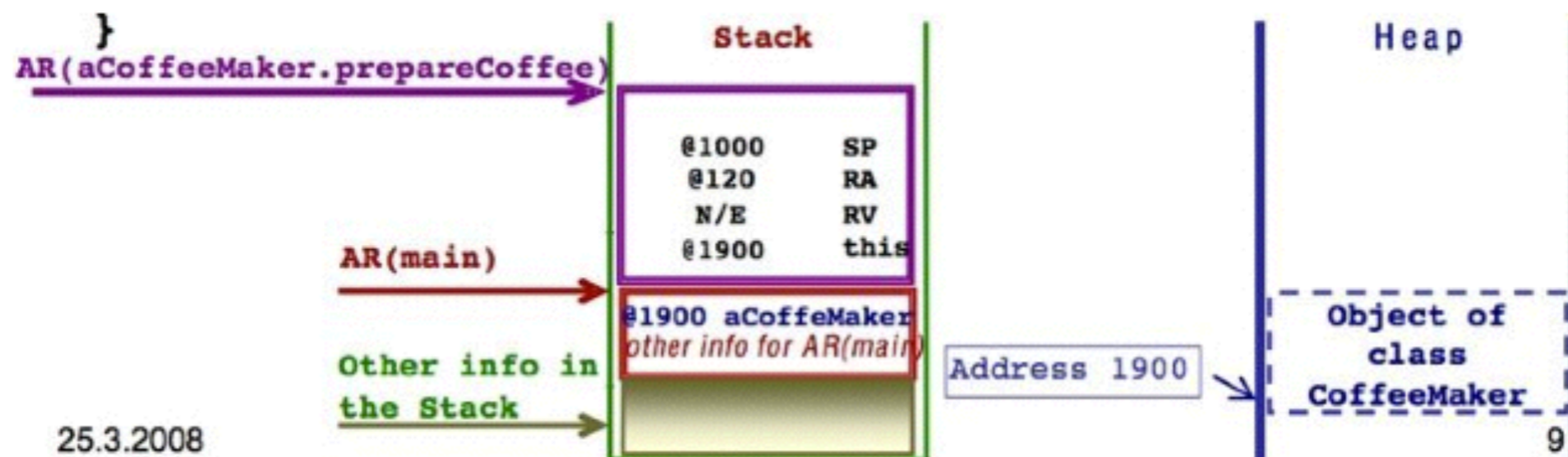
10

AR obsahuje

- Návratová hodnota (Co vracím?) **RV**
- Adresa návratu (Kam se vracím?) **RA**
- Ukazatel na zásobník (Odkud jsem přišel?) **SP**
- Hodnotu parametrů metody.
- Lokální proměnné.

Activation Record

```
public class CoffeeMaker {
    public void prepareCoffee() {
        /*Prepare the coffee with a standard amount of sugar*/
    }
    public void prepareCoffeeSweet(int sugarAm){
        /* Prepare the coffee with sugarAm(ount) of sugar */
    }
    void main(...) {
        CoffeeMaker aCoffeeMaker;
        aCoffeeMaker = new CoffeeMaker();
        aCoffeeMaker.prepareCoffee();
    }
}
```



25.3.2008

Převzato z [1]

Paměťový formalismus

- Umožňuje přehledně zaznamenat stav programu.
- Lze ho použít pro krokování programu “na papír”.

Hodnoty

- Logické: true, false
- Čísla: 0, 1, -1, 2, -2, ...
- Znaky: 'a', 'A', 'b', 'B', ...
- Řetězce: "Super řetězec", ...
- Adresy: null, #1, #2, #3, ...

Proměnné

- Jméno = hodnota
- např. contents = 5, nebo next = #3
- Hodnota proměnné musí odpovídat jejímu typu.

Objekty

- Třída(attr1 = hodn1, attr2 = hodn2, ...)
- např. Node(contents = 1, next = #2)
- Jména atributů musí korespondovat s danou třídou, na jejich pořadí nezáleží.
- Pokud atributy neznáme, můžeme je vypustit:
Node(...)

Halda

- Mapování z adres na objekty.
- Používáme šipkovou notaci (symbol není podstatný).
- #1 -> Node(contents = 2, next = #2),
#2 -> Node(contents = 3, next = #2)
- Na jedné adrese nemohou být dva objekty.

Prostor statických proměnných

- `Třída1.proměnná1 = hodnota1,`
`Třída2.proměnná2 = hodnota2`
- Např. `Node.globalhead = #2,`
`Car.totalCount = 10211432`
- Statické proměnné musí korespondovat s definicemi použitých tříd.

Zásobník

- Rámec = Stack Frame = Activation Record
- Zásobník se skládá z rámců, každý rámec odpovídá jedné zavolané metodě.
- Uvnitř rámce jsou vypsány existující lokální proměnné a jejich hodnoty.
- Rámce zpravidla píšeme nad sebe a oddělujeme vodorovnou čarou.
- Nesmíme zapomenout na this!

Zásobník - příklad

$a = 3, b = 5, \text{this} = \#1$

$x = \#1, a = \text{"abecede"}$

Celá paměť'

(halda | zásobník | statické proměnné)


```
class Coffee {
    public boolean milk;
    public Coffee() {
    }
}

public class CoffeeMaker {
    static int coffeeCount = 0;
    Coffee makeCoffee() {
        coffeeCount++;
        Coffee c = new Coffee();
        c.milk = true;
        return c;
    }
    public static void main(String[] args) {
        CoffeeMaker coffeeMaker;
        coffeeMaker = new CoffeeMaker();
        Coffee c = coffeeMaker.makeCoffee();
    }
}
```

Garbage Collection

Smysl Garbage Collection

- Manuální správa paměti může vést k chybám
 - úniky paměti (memory leak).
 - nesprávné ukazatele - ukazují na paměť, která již není alokována.
- GC umožňuje programátorovi se více soustředit na samotný program.

Garbage Collection

- Cíl: uvolnit objekty, které nebudou již více používány.
- **Nelze určit dopředu.**
- Dosažitelnost: tranzitivní uzávěr ukazatelů začínající v kořenové množině (root set) (všechny lokální a globální proměnné).

Metody GC

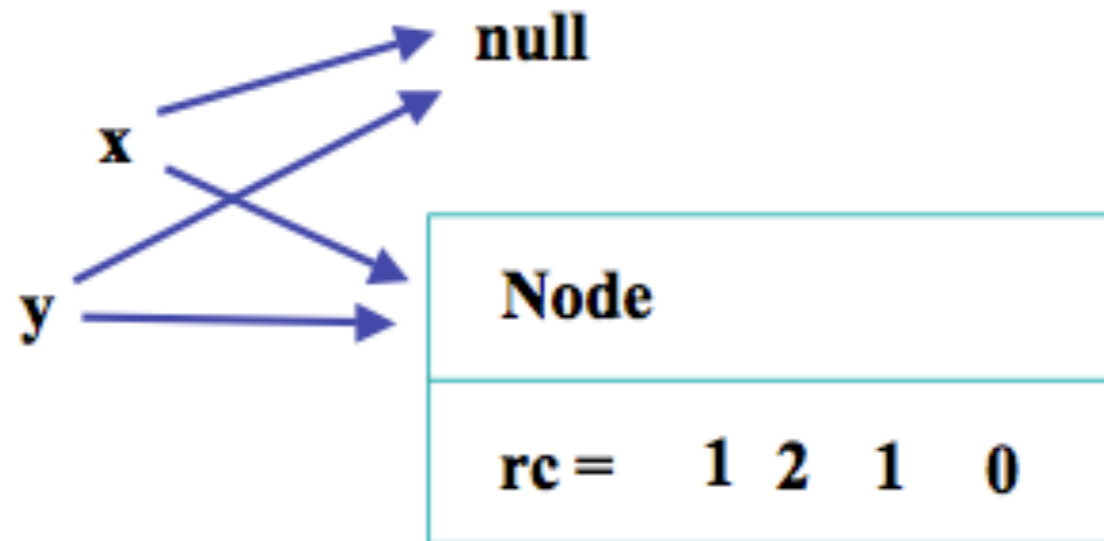
- Algoritmy
 - Reference Counting
 - Mark and Sweep, Mark and Compact, Mark and Copy
- Přístupy
 - Hybrid Collecting - kombinace předchozích
 - Generational Collectors (Java) - zavedení generací podle stáří objektů

Reference Counting

- Každý objekt obsahuje čítač.
- Přiřazení objektu do proměnné → snížení stavu čítače starého objektu a zvýšení stavu čítače objektu nově přiřazeného (pokud existují).
- Když čítač u objektu dosáhne 0, objekt se uvolní (jeho paměť připadne na free list).
- Nevýhody: velká přidaná zátěž, neporadí si s cyklickými odkazy.

Reference Counting

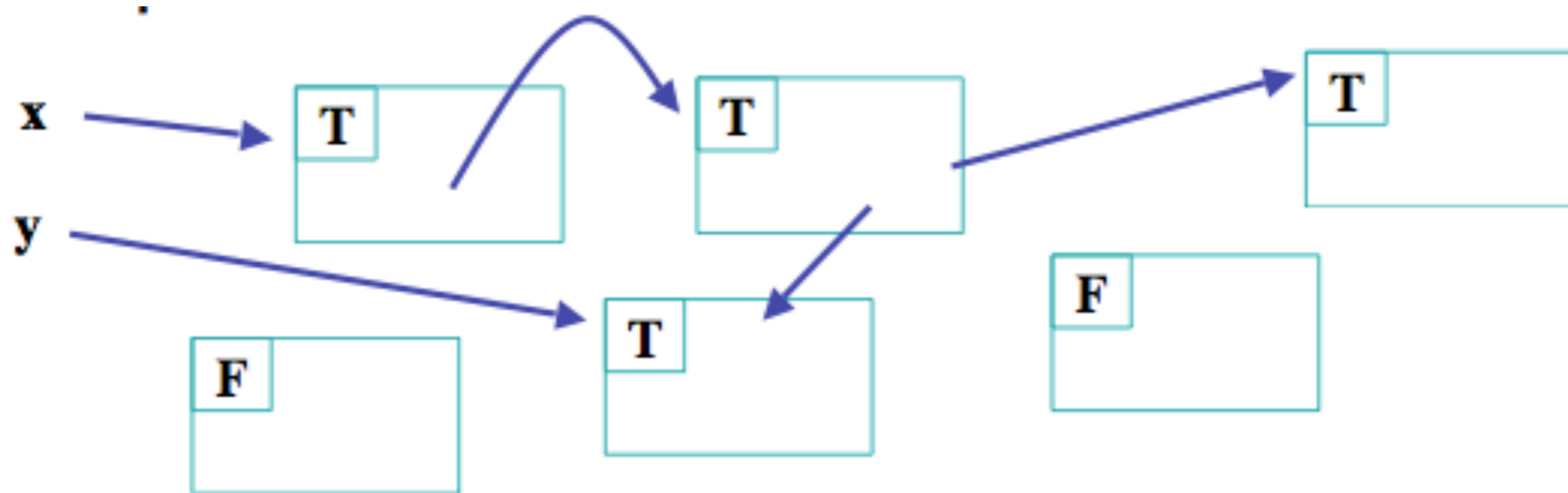
```
Node x, y;  
x = new Node (3, null);  
y = x;  
x = null;  
y = x;
```



Mark and Sweep

- Každý objekt obsahuje značku (mark).
- Jednou za čas dojde ke konstrukci transitivního uzávěru z kořenové množiny a všechny dosažitelné objekty jsou označeny.
- Neoznačené objekty jsou uvolněny (sweep)(přidání jejich paměti na free-list).
- Zbylé objekty jsou opět odznačeny.

Mark and Sweep



Reference

- [1] <http://task3.cc/wp-content/uploads/2011/04/object-oriented-memory-management-java-c++.pdf>