CS212

Java Practicum

Fall 2004
OOP

Announcements

- P3 due 11/12
- hopefully you didn't just start
- P4 due 12/3
- P4 currently being proofread...to be posted soon
- guest lecture next week...networking!

Overview

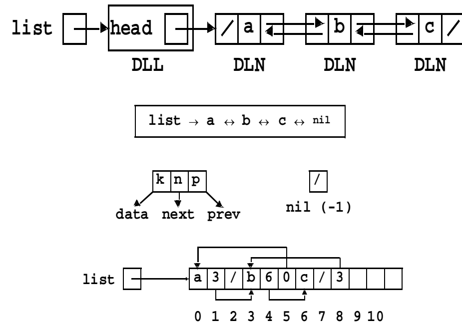
- OOP and Bali
- Pointers and Objects
- The Heap
- Samcode
  - Creating Objects
  - Instance Variables
  - Instance Methods
  - Object Literals

OOP and Bali

- How?
  - crazy syntax?
  - essentially like Java
- expressions and objects
  - what is a **`new`** ***`something()`***
  - ***`something.something`***
  - ***`something.something = something...`***
- need way to store and use value of **`new`** ***`something()`***
- how? carve out chunks of memory...

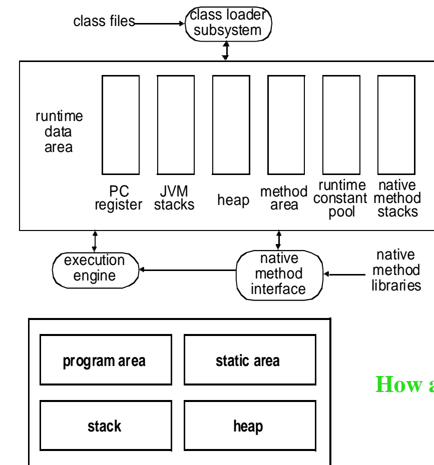
Objects and Pointers

An example:



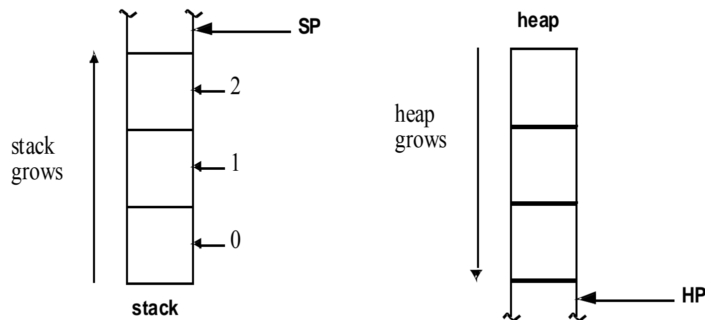
So, where do the objects in memory go....?

JVM and OOP



How about SaM?

SaM

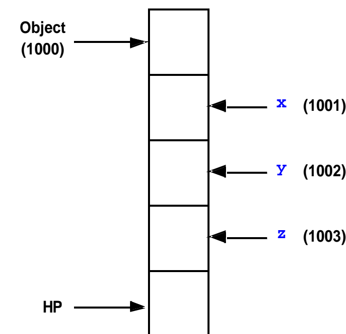


globals?
program area?

You've seen the stack...
So, what about the heap?

SaM's Heap

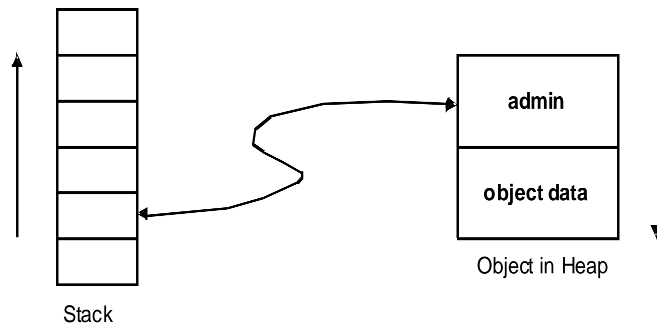
- Definition of object in SaM:
  - object = chunk of memory in heap
  - object = fields + 1 (or more)
  - address = beginning of chunk
  - no functions in Heap! (Stack keeps track)



- Object starts at first cell
- Fields stored in subsequent cells
- HP: next free cell in heap

More SaM's Heap

- Object connection to Stack?
  - address of object stored in stack (variable/param)
  - SaM's heap is currently same memory as stack (stack cells: 0—999, heap cells: 1000—?)



Samcode for Object

- Use memory allocation: **MALLOC**! (look familiar?)
  - pops top of stack
  - allocates that **number of cells + 1** in heap
  - pushes the address of the first cell onto stack
- Example (**OOP0.sam**):

```
PUSHIMM 1   // 1 cell to allocate
MALLOC      // pop 1 and allocate 1 cell in heap
PUSHIMM 3   // 3 cells to allocate
MALLOC      // pop 3 and allocate 3 cells in heap
PUSHIMM 0   // no cells to allocate
MALLOC      // pop 0 and allocate no cells in heap
ADDSP -3    // deallocate object addresses
PUSHIMM 0   // push dummy return value
STOP        // cease execution
```


Modeling Constructors

- *Constructor*: special kind of method that...
  - creates object
  - runs code
  - returns reference to newly created object
- The gist for creating object:
  - Bali:

```
{.... new Something(); ....}
```

  - Assume `Something` uses:

```
class Something {
    int x; int y;
}
```

  - SaM:

```
PUSHIMM 2 // 2 cells in object
MALLOC    // allocate 3 cells in heap and push address on stack
```

- How to run code and return reference?

Secret Parameter!

- Treat object as if were hidden parameter to a method
  - after calling **MALLOC**, reference to object sits on stack
  - why not use that reference as first parameter?
- Procedure (the gist):
  - allocate object (leaving object reference on stack)
  - start processing constructor as function, so
    - push other params
    - save old FBR, set new FBR
    - callee code (body of constructor):
      `JSR Classname_Classname`
    - code for constructor
    - store object address as rv
    - tear down frame
  - Since rv of constructor is object address, you can then save that value, as in `t = new Thing()`
- See Part 4 document for full procedure

Constructor Pattern

```
PUSHIMM 0
PUSHIMM f
MALLOC
code for other params
LINK
JSR Classname_Classname
UNLINK
ADDSP -(n+1)
Classname_Classname:
constructor Samcode
JUMP LabelEnd:
LabelEnd:
PUSHOFF -(n+1)
STOREOFF -(n+2)
ADDSP -locals
RST
```


Constructor Example

- Bali example:

```
int main( ) {
    Thing t ;
    t = new Thing( ) ;
    return 100 ;
}
public class Thing {
   public Thing( ) { }
}
```

- See **OOP1.sam**

Instance Variables

- ***Instance variables***:
  - the fields of a class
  - no static, global
- Need way to access object on heap
  - Could feasibly use HP just as SP?
  - Problem: need relative address!
- Solution:
  - use object address
  - so, can figure out field relative to that address (see figure on Slide 8)
  - but, *another* problem: no "Heap Based Register!"
    - so, use absolute address for object!
    - fields are still relative to that address
    - set offsets when compiling (like symbol table)

Absolute Addresses (one approach)

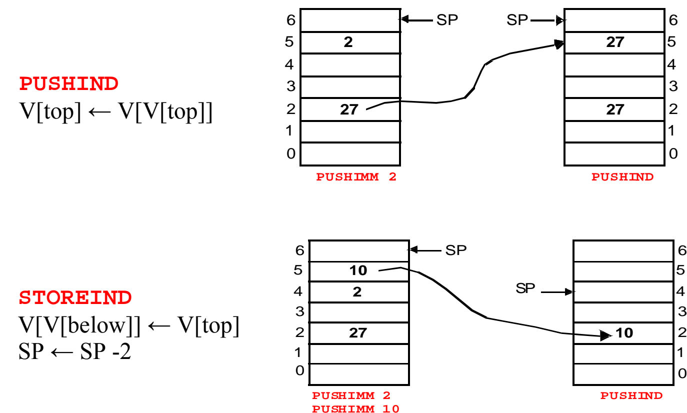
PUSHIND

V[top] ← V[V[top]]

STOREIND

V[V[below]] ← V[top]

SP ← SP -2



IV Procedures

- Store:

```
PUSHOFF address_of_object  // bottom of object in heap
PUSHIMM offset_of_field    // relative address of field
ADD                        // absolute address of field
PUSHIMM value_to_store     // push value to store in field
STOREIND                   // V[V[below]] <- V[top]
```

- Retrieve:

```
PUSHOFF address_of_object  // bottom of object in heap
PUSHIMM offset_of_field    // relative address of field
ADD                        // absolute address of field
PUSHIND                    // V[top] <- V[V[top]]
```


IV Example

- Bali:

```
int main( ) {
    Thing t ;
    t = new Thing ( 27 ) ;
    return t.x ;
}
public class Thing {
    public int x ;
    public Thing ( int v ) { x = v ; }
}
```

- See **OOP2.sam**

Instance Functions

- Very similar to contructors:
  - pass reference to object as first hidden parameter
  - Samcode follows pretty much same pattern as you have already seen
  - JSR labels like ***Classname_functionname***
- Design issues:
  - variable scope!
  - hidden object reference helps you access fields
  - but how to know if variable is a field?
    - symbol table lookup: if not in function scope, maybe belongs to object?
    - if not in object, maybe belongs to global?
    - if not global, it's crap!

Function Example

- Bali

```
int main( ) {
    Thing t ;
    int x ;
    x = 10 ;
    t = new Thing(x);
    return t.add(20);
}

public class Thing {
    public int x;
    public Thing(int v) { x=v; }
    public int add(int v) {
    return (v + x);
    }
}
```

- See **OOP3.sam**

Object Literals

- `this`
  - ***reference to current object***
  - so, need to pass object reference value
- `null`
  - no object
  - a bit trickier...convenient to use 0 address on stack
  - why? rv of program doesn't appear until end