COMP 520 - Compilers

Lecture 16 (March 29, 2016)

Runtime organization of object oriented languages

• Reading for today
  – PLPJ Chapter 6: secn 6.7

• Reading for next week
  – Review chapter 7
Today’s topics

• Review of miniJava classes without inheritance
  – mJAM representation of objects
  – location in memory

• mJAM virtual machine
  – execution of binary object code (example)
  – debugging capabilities

• mJAM support for classes with single inheritance
  – representation
  – mJAM support

• Related issues
miniJava: simple classes, no inheritance

- Classes

```java
class A { int x; void p() { x = 3; } }
```
- runtime entity descriptions in AST
  - class A: $S_A$ = size of class A (# fields)
  - field x: $d_x$ = displacement of field x
  - method p: $d_p$ = displacement of code for p

- Objects
  - objects are created on the heap: $A a = \text{new} A();$
  - let $d_a$ be displacement of local var “a” in activation record

![Diagram of object instance in heap and activation record on stack]
mJAM memory organization

- **Two separate memories**
  - **Code store**
    - compiler-generated program is loaded into code segment
    - predefined runtime functions are located in the primitive segment
    - TAM can not write into code store
  - **Data store**
    - static constants and variables are loaded into global segment
    - procedure invocation and expression evaluation uses execution stack
      - expands downwards
    - dynamically allocated values are allocated on the heap
      - expands upwards
      - memory for deleted values can be reused
- **ABI defines fixed addresses and usage conventions**
  - various locations in memories are accessed relative to machine registers (CB, SB, HT, etc.)
mJAM: runtime support for simple classes

- mJAM code sequences

A a = new A();  // (object creation)
a. x;          // (qualified reference)
a. p();        // (method invocation)
x = x + 3;      // (field upd within p())

LOADL -1
LOADL S_A
CALL newobj
STORE d_a[LB]

LOAD d_a[LB]
LOADL d_x
CALL fieldref

LOAD d_a[LB]
CALLI d_p[CB]

LOAD d_x[OB]
LOADL 3
CALL ADD
STORE d_x[OB]

“this”
call instance

OB [ within p() ]

object instance in heap
activation record on stack

miniJAvA runtime organization

COMP 520: Compilers - J. F. Prins
Classes with single inheritance (Java)

- **Class hierarchy**
  
  ```java
  class A { int x; void p() { ... } }
  class B extends A { int y; void p() { ... } void q() { ... } }
  ```

  - **inheritance hierarchy**
    - “class B extends class A”, or “B is a subtype of A”

  - **fields**
    - fields of B extend the fields of A
    - runtime layout of fields in A is a prefix of the runtime layout of fields in B

  - **methods**
    - methods of B extend the methods of A
    - methods of B can redefine (override) methods of A
Static and dynamic type with single inheritance

- **Object type**
  - **static type (declared type)**
    - used by compiler for type checking
    - determines accessible fields and available methods on objects
    - type rules for assignments
      - assignment: type of RHS must be a subtype of type of LHS
      - method call: type of arg $i$ must be a subtype of type of parameter $i$
  - **dynamic type (run-time type)**
    - generally only known at runtime
      - part of the representation of an object
        - initialized at time of creation from object constructor
      - dynamic type is always a subtype of the static type (guaranteed by type system)
      - dynamic type determines which method is invoked (method override)

- **examples**
  ```java
  A a = new A();
  B b = new B();
  A c = b;
  B d = a;
  a.p();
  b.q();
  c.p();
  ```
  ```java
  class A {
    int x;
    void p(){ ... }
  }
  ```
  ```java
  class B extends A {
    int y;
    void p(){ ... }
    void q(){ ... }
  }
  ```
mJAM representation of single inheritance

```
class A { int x; void p() { ... } }
class B extends A {
    int y;
    void p() { ... }
    void q() { ... }
}
```

- runtime entity descriptions in AST
  - class A: \( S_A = \) size of class A
  - class A: \( d_A = \) displacement of class descriptor for A
  - class B: \( S_B = \) size of class B (including size of class A)
  - class B: \( d_B = \) displacement of class descriptor for B
  - field x: \( d_x = \) displacement of field x in A and B
  - field y: \( d_y = \) displacement of field y in B
  - method p: \( h_p = \) index of method p in A and B
  - method q: \( h_q = \) index of method q in B
  - method p in A: \( d_{p[A]} = \) displacement of code for p() in A
  - method p in B: \( d_{p[B]} = \) displacement of code for p() in B
  - method q in B: \( d_{q[B]} = \) displacement of code for q() in B
Classes with single inheritance

- mJAM runtime layout
Classes with single inheritance

- mJAM code sequences (only changed sequences are shown)

```java
A a = new A();
// (object creation)
LOADL dA
LOADL SA
CALL newobj
STORE da[LB]

a.p();
// (dynamic invocation)
LOAD da[LB]
CALLD hp
```

```
<table>
<thead>
<tr>
<th>SB</th>
</tr>
</thead>
<tbody>
<tr>
<td>dA</td>
</tr>
<tr>
<td>dB</td>
</tr>
<tr>
<td>hq</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p:</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp[A]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>q:</th>
</tr>
</thead>
<tbody>
<tr>
<td>dq[B]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LB</th>
</tr>
</thead>
<tbody>
<tr>
<td>dB</td>
</tr>
<tr>
<td>SB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a:</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b:</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
</tbody>
</table>

| dA | SA |

| x |

| dA | x |

| dB | SB |

| x |

| dy |

| dx |
```
Related issues

• single inheritance
  – type operations
    • instanceof
    • casting
  – super() superclass constructor invocation

• multiple inheritance
  – we lose the prefix property of runtime layout!

• optimization
  – dynamic method dispatch has high cost
  – converting dynamic to static calls

• dynamically loaded classes
  – Java loads classes on demand, hence cannot use “global” optimization techniques