5th Practical Exercise
Translation of OO Concepts to MIPS

Lecture Compilers SS 2011

Jan Schäfer

Software Technology Group
TU Kaiserslautern

MiniJava^{++}

```
class B {
  int x = 5;
  int getX() { return this.x; }
}

class DoubleB extends B {
  int y = 6;
  int getX() { return this.x*2; }
  int getY() { return this.y; }
}

class Main {
  static void main() {
    B b = new B();
    b = new DoubleB();
    int z = b.getX();
  }
}
```

MiniJava^{++}

```
class B {
   int x = 5;
   int getX() { return this.x; }
}

class DoubleB extends B {
   int y = 6;
   int getX() { return this.x*2; }
   int getY() { return this.y; }
}

class Main {
   static void main() {
      B b = new B();
      b = new DoubleB();
      int z = b.getX();
   }
}
```

MiniJava^{++}

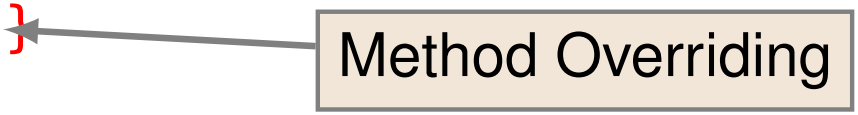
```
class B {
   int x = 5;
   int getX() { return this.x; }
}

class DoubleB extends B {
   int y = 6;
   int getX() { return this.x*2; }
   int getY() { return this.y; }
}

class Main {
   static void main() {
      B b = new B();
      b = new DoubleB();
      int z = b.getX();
   }
}
```

MiniJava^{++}

```
class B {
   int x = 5;
   int getX() { return this.x; }
}

class DoubleB extends B {
   int y = 6;
   int getX() { return this.x*2; }
   int getY() { return this.y; }
}

class Main {
   static void main() {
      B b = new B();
      b = new DoubleB();
      int z = b.getX();
   }
}
```

MiniJava^{++}

```
class B {
   int x = 5;
   int getX() { return this.x; }
}

class DoubleB extends B {
   int y = 6;
   int getX() { return this.x*2; }
   int getY() { return this.y; }
}

class Main {
   static void main() {
      B b = new B();
      b = new DoubleB();
      int z = b.getX();
   }
}
```



MiniJava^{++}

```
class B {
   int x = 5;
   int getX() { return this.x; }
}

class DoubleB extends B {
   int y = 6;
   int getX() { return this.x*2; }
   int getY() { return this.y; }
}

class Main {
   static void main() {
      B b = new B();
      b = new DoubleB();
      int z = b.getX();
   }
}
```

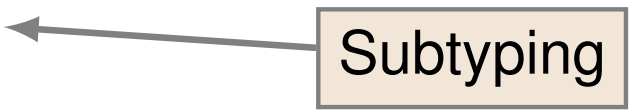
Class Types

MiniJava^{++}

```
class B {
   int x = 5;
   int getX() { return this.x; }
}

class DoubleB extends B {
   int y = 6;
   int getX() { return this.x*2; }
   int getY() { return this.y; }
}

class Main {
   static void main() {
      B b = new B();
      b = new DoubleB();
      int z = b.getX();
   }
}
```

MiniJava^{++}

```
class B {
   int x = 5;
   int getX() { return this.x; }
}

class DoubleB extends B {
   int y = 6;
   int getX() { return this.x*2; }
   int getY() { return this.y; }
}

class Main {
   static void main() {
      B b = new B();
      b = new DoubleB();
      int z = b.getX();
   }
}
```



MiniJava^{++}

```
class B {
   int x = 5;
   int getX() { return this.x; }
}

class DoubleB extends B {
   int y = 6;
   int getX() { return this.x*2; }
   int getY() { return this.y; }
}

class Main {
   static void main() {
      B b = new B();
      b = new DoubleB();
      int z = b.getX();
   }
}
```

Grammar Extension

MiniJava.cup

```
Primary ::= ...
  | NEW IDENTIFIER
  | NULL ;

ClassDeclaration ::= ...
  | CLASS:c IDENTIFIER:id EXTENDS IDENTIFIER:sid ClassBody:body ;

Type ::= ...
    | IDENTIFIER ;
```

AST Extension

MiniJava.katja

```
Expression = ...
           | Null ( SourcePosition sourcePos )
           | New ( SourcePosition sourcePos, Identifier ident )

Type = ...
     | Identifier

IdentifierOpt = IdentifierNone ()
              | Identifier

ClassDecl ( SourcePosition sourcePos, Identifier ident,
            IdentifierOpt superClass,
            ClassBodyDecls body )
```

MIPS-AST Extension

Jump and link register

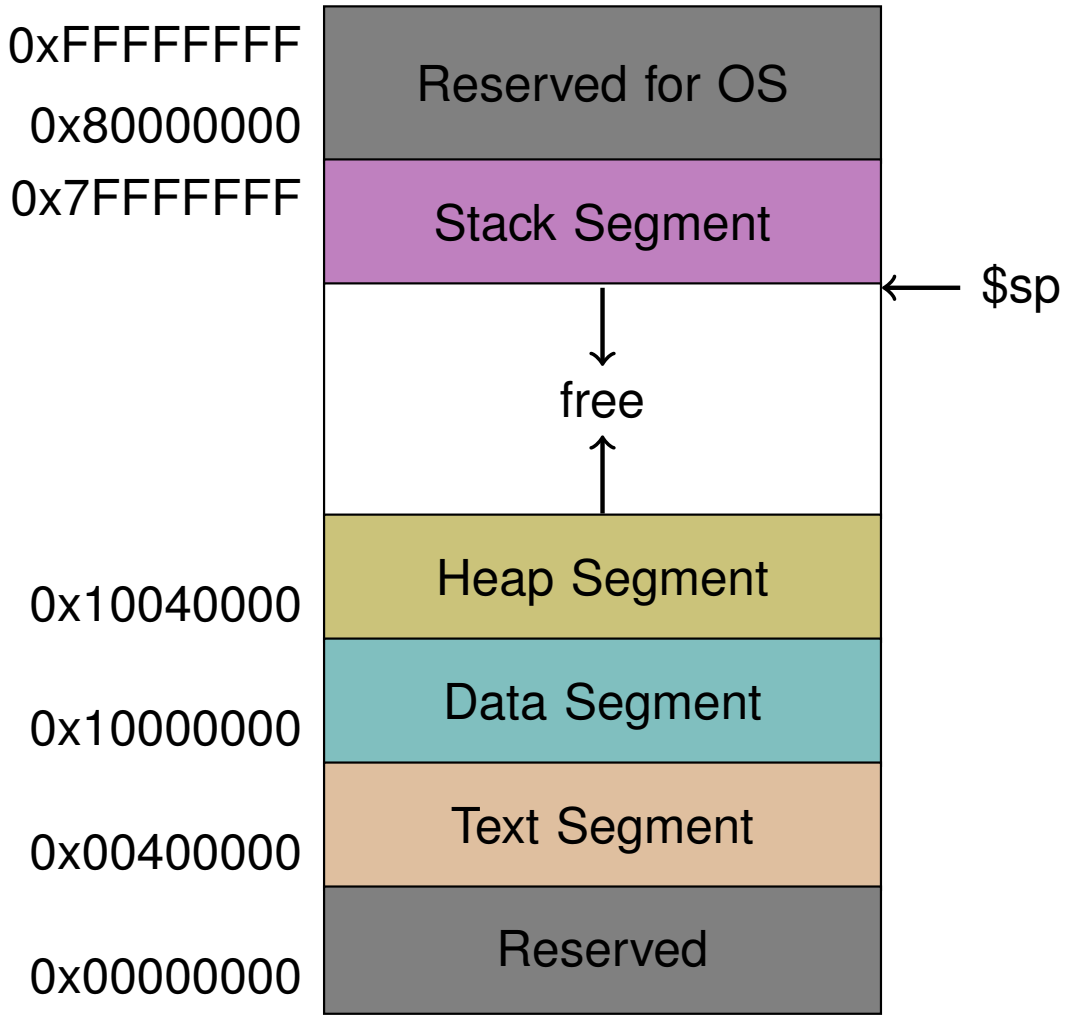
```
Instruction = ...
            | JALR (Register reg0)
```

Type System Extension

- Class names are types
- `Object` as implicit super class
- Handle subtyping in
  - Assignments (left <: right)
  - Method calls (argument <: parameter)
  - Returns (return value <: return type)
- `null` <: all class types
- Optional: Check correct overriding of methods
  - Optimal: Covariant return types, contravariant parameter types
- Optional: No cycles in type hierarchy

Object Allocation

MIPS Memory Structure



Allocate Heap Memory in MIPS/MARS

- Heap starts at 0x10040000 in MARS
- Store free pointer in global variable
- Allocate memory on heap:

```
.data
free: .word 0x10040000  # start at top of heap

.text
lw   $t0, free      # get next free memory address
move $s0, $t0       # remember memory address in $s0
addi $t0, $t0, 12   # allocate 12 bytes of memory
sw   $t0, free      # update free pointer
sw   $zero, 0($s0)  # initialize free memory
sw   $zero, 4($s0)  # initialize free memory
sw   $zero, 8($s0)  # initialize free memory
```

Alternative Allocation using a Syscall

MARS provides a syscall for allocating heap space

```
li $a0, 12           # allocate 12 bytes of memory
li $v0, 9            # memory allocation syscall
syscall
move $s0, $v0        # $v0 contains address of allocated memory
sw   $zero, 0($s0)   # initialize free memory
sw   $zero, 4($s0)   # initialize free memory
sw   $zero, 8($s0)   # initialize free memory
```

Object Allocation

Memory Management

- Divide heap into used and unused part
- Global variable `free` points to unused heap
- `free` is increased for each object allocation
- No garbage collection

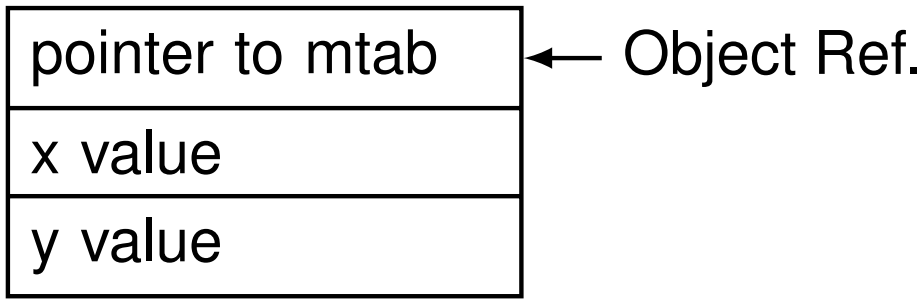
Object Allocation (2)

Object Representation

- Pointer to virtual method table
- Space for each field of the object

Example

```
class A {
  int x;
}
class B extends A {
  int y;
}
```



Dynamic Binding

Virtual Method Table

Example

```
class A {
   int getX() { ... }
}

class B extends A {
   int getX() { ... }
   int getY() { ... }
}
```

Virtual Method Table

Example

```
class A {
  int getX() { ... }
}

class B extends A {
  int getX() { ... }
  int getY() { ... }
}
```

```
.data
A_mtab: .word A_getX
B_mtab: .word B_getX, B_getY

.text
A_getX: ...
B_getX: ...
B_getY: ...
...
```

Dynamic Call

```
A a = ...
a.getX();
```

```
...                 # assume a is stored in $t0
lw   $t1, 0($t0)    # load address of mtab into $t1
lw   $t2, 0($t1)    # get address of method getX (first method)
move $a0, $t0       # pass implicit parameter as argument
jalr $t2            # call method
```