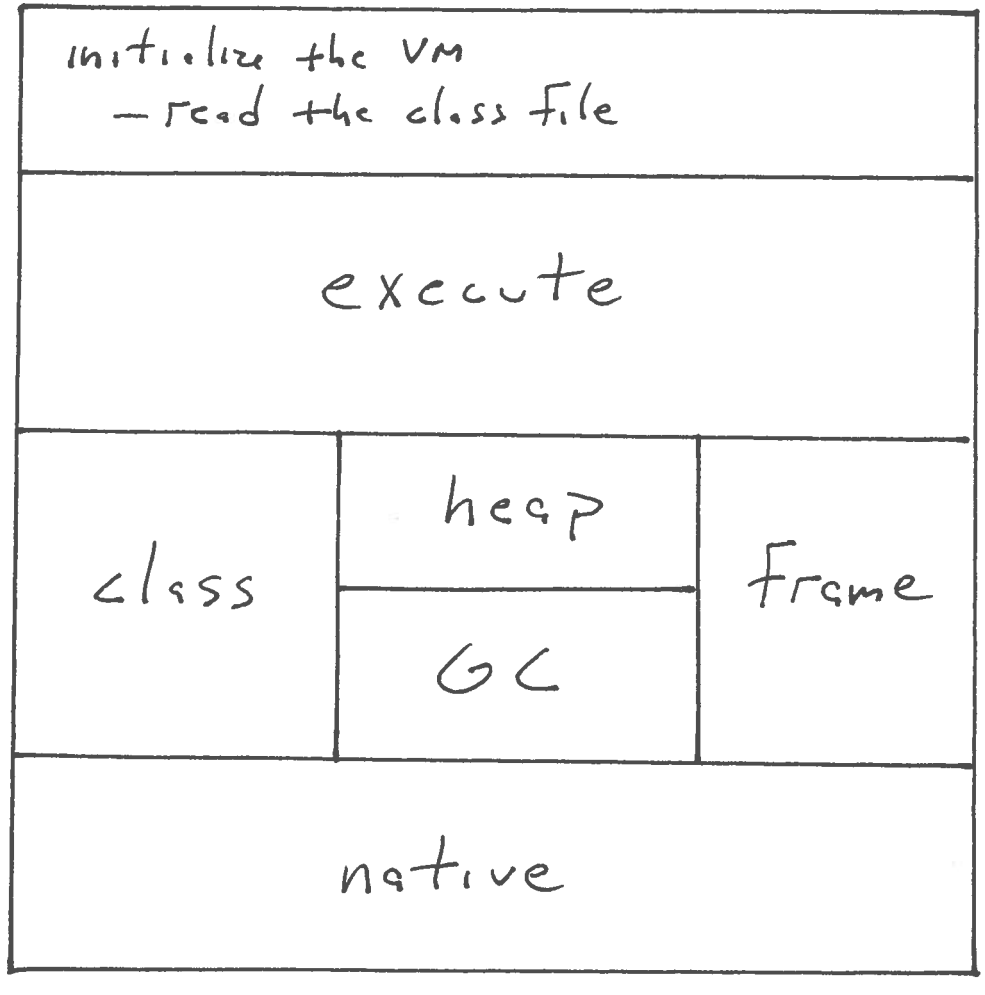
Implementing the maTe Virtual Machine

CS520

Dept. of Computer Science
Univ. of New Hampshire

reading the class file is step #1 in implementing the mcTe virtual machine

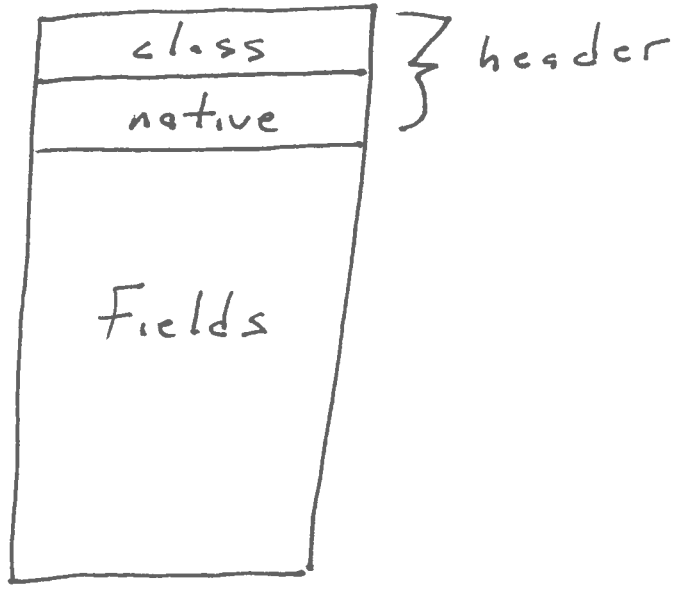


heap

where objects are stored

the key issue:

object layout



each object will have a two-word header

1. address of class descriptor
2. reserved for use by native objects
   e.g. the value of an Integer

note: in object-oriented languages, a run-time type is maintained

↳ that is, the interpretation of the bits in the object is added as a tag to the object.

↳ this allows the same code to have different behavior based upon the tag

ie virtual method call

```
// abstract class
class Animal {
  String noise() { return "<abstract>"; }
}

class Pig extends Animal {
  String noise() { return "oink"; }
}

class Dog extends Animal {
  String noise() { return "woof"; }
}

Integer main()
{
  Animal x;
  String y;

  y = in;
  if (y.toInteger().equals(42))
  {
    x = new Pig();
  }
  else
  {
    x = new Dog();
  }

  out x.noise();
  out newline;

  return 0;
}
```

```
Animal:
  $Object
  0
  4
  0 1 "Object$equals$Object"
  0 2 "Object$hashCode"
  0 3 "Object$toString"
  $Animal$noise 1 "Animal$noise"
Pig:
  $Animal
  0
  4
  0 1 "Object$equals$Object"
  0 2 "Object$hashCode"
  0 3 "Object$toString"
  $Pig$noise 1 "Pig$noise"
Dog:
  $Animal
  0
  4
  0 1 "Object$equals$Object"
  0 2 "Object$hashCode"
  0 3 "Object$toString"
  $Dog$noise 1 "Dog$noise"


mainBlock:
  aconst_null
  astore 0
  aconst_null
  astore 1
  in
  astore 1
  aload 1
  invokevirtual 6 1
  newint 42
  invokevirtual 0 2
  ifeq $L.0
  new $Pig
  dup
  invokespecial $Pig_constructor 1 1
  astore 0
  goto $L.1
L.0:
  new $Dog
  dup
  invokespecial $Dog_constructor 1 1
  astore 0
L.1:
  aload 0
  invokevirtual 3 1
  out
  newstr "
"
  out
  newint 0
  goto $mainBlock$.exit
  newint 0
mainBlock$.exit:
  areturn
```

% mc5520/bin/mc < animals.m > animals.s

% mc5520/bin/mas < animals.s > animals.class

Support For Garbage Collection

two bits in the header must be given to the GC

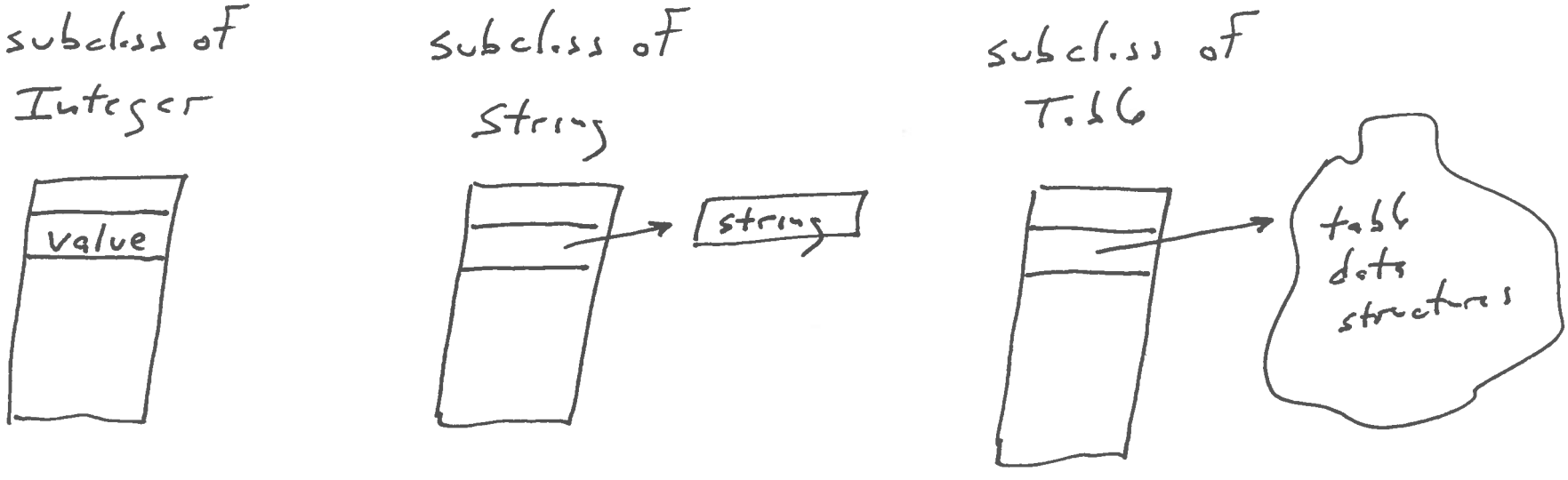
1. indicate block as free or allocated
2. used during GC's marking phase

low order two bits of class descriptor can be used since addresses must be mult. of 4

GC also needs to know length of blocks but that can be computed from the class descriptor

i.e. (# fields + 2) * 4 bytes

2nd header word



"secondary" objects will need to have headers to support the GC

heap functions

allocateObject(class)

get #fields from class
allocate block containing #fields + 2 words
put class in first word
return base address of block

allocateString(initialValue)

like allocateObject

but need to also allocate space for the string

set the 2nd word of the object to point to a copy of initialValue

getclass(ref)

ref points to the object's block

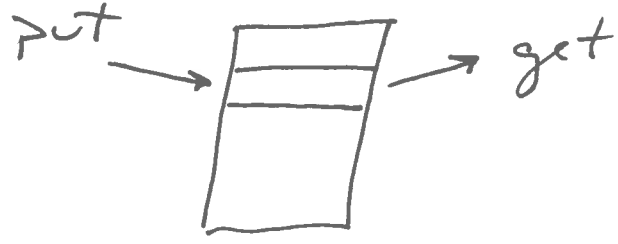
return its first word

getStringValue(ref)

return the 2nd word

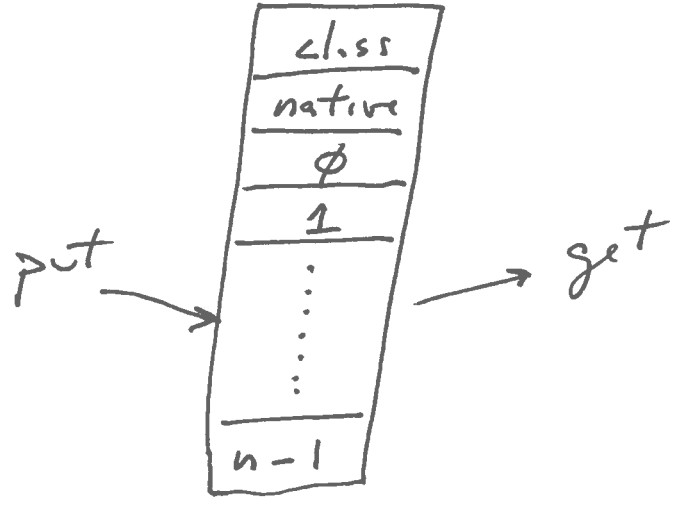
getInteger / putInteger

read/write the 2nd word



getField / putField

read/write the word containing a field



Frames

there is a stack of Frames

initially Frame created for the main block

when method is called, new Frame created and pushed on the stack

when method returns, Frame is popped from stack and deleted

what is in a frame?

local variables

operand stack

saved PC (i.e. the return address)

local variables varies

```
////////////////////////////////////////////////////////////////////////
// support for frames
//    a frame contains a local slot array, a PC, and an operand stack

// handle for frames
typedef void *Frame;

// create a frame given its local slot array length and its initial PC
Frame createFrame(Count localsLength, Address pc);

// delete a frame
void deleteFrame(Frame frame);

// is operand stack in frame empty?
bool isEmptyOperandStack(Frame frame);

// push reference onto operand stack of frame
void pushOperandStack(Frame frame, Reference ref);

// pop reference from operand stack of frame
Reference popOperandStack(Frame frame);

// retrieve the i-th item on the operand stack of frame (0 is the top,
// 1 is one beneath the top, 2 is two beneath the top, etc)
Reference peekOperandStack(Frame frame, Index i);

// return length of local slot array of frame
Count lengthLocals(Frame frame);

// get reference from local slot array of frame given its index
Reference getLocals(Frame frame, Index index);

// put reference into local slot array of frame given its index
void putLocals(Frame frame, Index index, Reference ref);

// get PC from frame
Address getPC(Frame frame);

// put PC into frame
void putPC(Frame frame, Address pc);

////////////////////////////////////////////////////////////////////////
// support for the frame stack
//    note: the frame stack (and there is only one) is created by
//          initializeVM.

// is frame stack empty?
bool isEmptyFrameStack(void);

// push frame onto frame stack
void pushFrame(Frame frame);

// pop frame from frame stack
Frame popFrame(void);

// get top frame from frame stack but do not pop
Frame getTopFrame(void);
```

the Fetch/execute cycle

initiated from the main function

```
initializeVM
create and push a frame
      for the main block
pc = getMainBlockAddress
while (1) {
      opcode = getWordFromCode(pc)
      pc += 4
      pc = execute(opcode, pc)
}
```

execute

switch on opcode

implement the particular instruction

may need to fetch additional words
using the pc

may need to update the pc

aconst_null

```
pushOperandStack(getTopFrame(), ∅);
```

aload

```
frame = getTopFrame();
index = getWordFromCode(pc);
pushOperandStack(frame, getLocals(frame, index));
return pc+4;
```

areturn

pop a frame
pop the return value from the operand
    stack in that frame
set the pc from that frame
is this a return from the main block?
    is frame stack now empty?
    if so, call haltVM with integer
        value from the return value

otherwise
    push return value onto operand stack
        of frame on top of the frame stack
    return the pc
    free the popped frame too