

More fun with Java

The Basics

Basic Structure of a Java Program

Packages

Simple Declarations

Primitive Types

Arrays

Classes

Class

Method

Constructors

Control

Iteration

Conditional Expressions

Exceptions

Tools

Basic Structure of a Java Program

```
/* a comment */  
// a comment
```

<package declaration>

<import declarations>

<class definitions>

Packages

```
package mit16_410_13.search.uninformed;
```

Package structure matches directory structure

```
mit16_410_13  
  search  
    uninformed
```

Package must be imported before they
can be accessed in a program

Import

```
import mit16_410_13.search.uninformed.Myclass;
```

```
import mit16_410_13.search.uninformed.*;
```

Simple Declarations

Primitive Types

```
int x, y, // uninitialized variables  
    z=511, h=0x1FF, oc=0777; // decimal, hex, octal  
double pi=3.14159265358979323846, ff=6.28e-3;  
boolean flag=true, done=false;  
char sep=' ';  
string myfirstname="paul", myfamilyname="robertson",  
        myfullname=myfirstname+" "+myfamilyname;
```

Arrays

Array Variables

```
int[] ai; // an array of int
char[][] aac; // an array of array of char
Object[] ao1, ao2; // two arrays of objects
int num, nums[], nums[][];
```

Creating Arrays

```
int[] ai={1, 2, 3}
double[] ad=new double[57];
int[][] aai={{1, 2}, {3, 4}, null};
```

Basic statement structure

```
x=3;
{ x=3; y=4; z=6; }
{ int x=2; y=x+1; z=y+2; }
```

Iteration

Do

```
do <statement> while (<expression>);

do {
    promptforusername();
}
while (readusername());
```

Iteration

While

```
while (<expression>) do <statement>;

while (node!=goalnode) do {
    node=node.next();
}
```

Iteration

for

```
for (<initexp>opt; <expression>opt; <update>opt) <stmt>

for (int i=1; i<10; i++) System.out.println(i);

for (int i=1; i<10; i++) {
    if (g(i)==0) continue; // goto innermost enclosing
                          // while do or for
    system.out.println(i);
    if (f(i)==0) break;
}
```

Break and labeled statement

```
foo: {
    dothis();
    dothat();
    if (imthrough()) break foo;
    domore();
}
```

Conditional Expressions

```
if
if (<expression>) <statement>
if (<expression>) <statement> else <statement>

if (x>3) foo(x);
if (y<2) goo(y);
if (z==2) foo(x) else goo(y);
if (z==2) if (y<2) goo(y); else bar(z); //error
```

Conditional Expressions

```
Switch
switch (<expression>) <switchblock>

switch (i) {
  case 0: something();
  case 1: somethingelse();
           break;
  default: handleothercases();
}
```

Classes

Class

```
class <classname> [implements interface1, interface2 ,...]
                  [extends <classname>]
{
  <declarations>
}
```

Method

```
public int myMethod (int limit) {
  ...;
}
```

Constructors

```
public MyClass (int arg) {
  super(arg);
  this.arg = arg;
}
```

Exceptions

Catching an exception

```
try <block> <catches>
try <block> <catches>opt <finally>
```

```
try {
  ... Some code ...
}
catch (NullPointerException e) {
  System.out.println("oops");
}
catch (Exception e) { ... }
finally { ... }
```

Exceptions

Raising an exception

```
throw <expression>;
// expression must be of type throwable

throw new MyException("Hello!");
```

Exceptions

Defining new exception

```
Class MyException extends Exception {
  MyException() { super(); }
  MyException(String s) { super(s); }
}
```

Exceptions

Methods that raise exceptions

```
public int myMethod (int limit) throws MyException
{
    ...
    if (badcondition()) throw new MyException("Oh Foo");
    ...
}
```

Some Predefined Exceptions

ClassNotFoundException
IllegalAccessException
ClassCastException
ArithmeticException
ArrayStoreException
IndexOutOfBoundsException
NullPointerException
NegativeArraySizeException

Tools

To run a compiled java program

```
java <class> arg_0 arg_1 ... arg_n
```

To compile a program

```
javac [-g] <filename> (Must include .java extension)
```

To debug a program

```
jdb <class> arg_0 arg_1 ... arg_n
```

jdb Commands

```
run [class [args]] -- start execution of application's main class
```

```
threads [threadgroup] -- list threads
thread <thread id> -- set default thread
suspend [thread id(s)] -- suspend threads (default: all)
resume [thread id(s)] -- resume threads (default: all)
where [thread id] | all -- dump a thread's stack
wherei [thread id] | all -- dump a thread's stack, with pc info
up [n frames] -- move up a thread's stack
down [n frames] -- move down a thread's stack
kill <thread> <expr> -- kill a thread with the given exception object
interrupt <thread> -- interrupt a thread
```

```
print <expr> -- print value of expression
dump <expr> -- print all object information
eval <expr> -- evaluate expression (same as print)
set <lvalue> = <expr> -- assign new value to field/variable/array element
locals -- print all local variables in current stack frame
```

```
classes -- list currently known classes
class <class id> -- show details of named class
methods <class id> -- list a class's methods
fields <class id> -- list a class's fields
```

```
threadgroups -- list threadgroups
threadgroup <name> -- set current threadgroup
```

```
stop in <class id>.<method>[(argument_type,...)]
-- set a breakpoint in a method
stop at <class id>:<line> -- set a breakpoint at a line
clear <class id>.<method>[(argument_type,...)]
-- clear a breakpoint in a method
clear <class id>:<line> -- clear a breakpoint at a line
clear -- list breakpoints
catch <class id> -- break when specified exception thrown
ignore <class id> -- cancel 'catch' for the specified exception
watch [access[all]] <class id>.<field name>
-- watch access/modifications to a field
unwatch [access[all]] <class id>.<field name>
-- discontinue watching access/modifications to
```

```
trace methods [thread] -- trace method entry and exit
untrace methods [thread] -- stop tracing method entry and exit
step -- execute current line
step up -- execute until the current method returns to
ler
stepi -- execute current instruction
next -- step one line (step OVER calls)
cont -- continue execution from breakpoint

list [line number[method]] -- print source code
use (or sourcepath) [source file path]
-- display or change the source path
exclude [class id ... | "none"]
-- do not report step or method events for spec
lasses
classpath -- print classpath info from target VM
```

```
monitor <command> -- execute command each time the program
stops
monitor -- list monitors
unmonitor <monitor#> -- delete a monitor
read <filename> -- read and execute a command file

lock <expr> -- print lock info for an object
threadlocks [thread id] -- print lock info for a thread

pop -- pop the stack through and including the curr
me
reenter -- same as pop, but current frame is reentered
redefine <class id> <class file name>
-- redefine the code for a class
```

```
disablegc <expr> -- prevent garbage collection of an object
enablegc <expr> -- permit garbage collection of an object

!! -- repeat last command
<n> <command> -- repeat command n times
help (or ?) -- list commands
version -- print version information
exit (or quit) -- exit debugger
```