Data Types, Decisions, And Iteration

Identifier

- a name for a
  - variable: camelCase
  - parameter: camelCase
  - constant: CASE
  - user-defined method/class: case
- case-sensitive
- has a *type* associated with it
- introduced with declaration of type

Type

- primitive: stored directly
- object: stored as a reference
- can be casted to another compatible type
  - if truncate, must explicitly state/declare

Integers

- four bytes
- stored exactly as binary digits (where one digit stores positive/negative sign)
- max value: $2^{31} - 1$
- min value: -2^{31}

Double

- eight bytes
- are stored in two parts
  - mantissa specifies the digits (one stores the sign)
  - exponent
- when converted to binary, there is roundoff error

Boolean

- true or false

Operators

- arithmetic + - * / %
  - applied to int and doubles
  - if involves both, int is promoted to double
  - PEMDAS applies
- relational == != > < >= <=
  - evaluate to true/false
  - generally used for primitive types
    - use methods such as compareTo and equals for objects
- logical ! && ||
  - applied to boolean expressions to compound them
  - short circuit evaluation
    - stops when has definite value
- assignment = += -+ *= /+ %=

chaining next=prev=sum=0;
evaluates right to left
increment/decrement ++ --

precedence

highest	! ++ --
	* / %
	+ -
	< > <= >=
	== !=
	&&
	||
lowest	= += -= *= /= %=

Control Structures

make the statements run in non-sequential order
Decision Making vs. Iteration

Decision Making

what path
if statement
if else statement
nested if statement
else gets matched with the closest if statement (unless brackets group differently)
extended if statement
else if does not exist
else
if

Iteration

repeat
for loop
for (initialization; termination condition; update statement)
statements
1) initialize
2) terminate condition
    a) if false, terminate
3) statements
4) update statement
5) repeat 2-5
loop variable should not have value changed inside loop body

scope of loop variable restricted to loop

for each loop

for (SomeType element: collection)

statements

“for each element of type SomeType in collection”

can not replace/remove elements

can modify instance fields

while loop

“if evaluates to true, execute, repeat”

statements must somehow lead to termination

do while loop

mr. horn does not recommend

nested loop

Classes and Objects

Objects

- a single instance of a class
- an idea represented as an object reference
- characterized by its
  - state
    - instance variables
  - behavior
    - methods

Classes

- a software blueprint for implementing objects
- maintains state and provides behavior
  - instance fields
  - constructor
  - accessor/ mutator methods

Keywords (access specifiers)

- public
  - usable by all client programs
- private
  - only accessed by some (ex: package)
- static
  - shared by all instances
- final > variables
  - can not be changed

Methods

- Headers
  - access specifier | return type | method name | parameter list
- Constructors
  - creates an object of a class
  - default
    - has no arguments
    - provides reasonable initial values
  - with parameters
    - sets instance variables using arguments
  - always same name as class
- Accessors
  - accesses class object without altering
  - suppose to return info of object
  - usually has no parameters
- Mutators
  - modifies at least one instance field
  - doesn't usually return anything
  - usually has parameters

Static Class

performs operation on entire class, not individual object
can't call instance method
can use static variables
ex: math class
(usually) does not create any objects
when declared, has no implicit parameter (no object)

References and Variables

stored as an address (reference) in memory of object
aliasing
can have multiple references to same object

Scope

of a variable or method is the region it can be accessed
hierarchy

CLASS

instance fields

static variables

METHODS

BLOCK

local variable

local variables take precedence over instance variables
unless you use the implicit parameter *this*

Parameters

information passed to the program
formal parameters are placeholders for actual parameters called *arguments*
there are explicit parameters and one implicit parameter (the object)
parameters are passed by value

any changes made to the parameters will NOT affect the values of the arguments called

methods can't change object referenced
but can change state

Inheritance and Polymorphism

Inheritance

defines a relationship between objects that share characteristics

Subclass extends Superclass

inherits characteristics (variables and methods) of superclass

can not directly access anything private

can override superclass methods by defining a method with same

- name
- return type
- parameters

can partial override methods by defining a method with

- call to super.method();
- additional statements

if subclass has no constructor, generates superclass default constructor

- *if none, compile time error*

can call super() in constructor

- *if and only if its the first line*

more rules on page 133 of barrons book

can have a superclass reference for a subclass object

Polymorphism

selecting the appropriate method for a particular object in class hierarchy

dynamic binding

- run time decision of which instance method to call

ex:

```
Superclass a = new Subclass( );
a.method(b);

at compile time
        reference a must have method called
        b must be the correct type
at runtime
        actual object's method will be called
```

Abstract Classes

can not be instantiated

an object is application specific, but

incomplete without subclass

classes *extend* abstract classes

some implementation

can have constructors, instance variables,
public static fields, constants

a class

abstract methods must be declared abstract

does not necessarily have to have abstract
methods

Interfaces

can not be instantiated

methods are equally applicable in variety of programs

classes *implement* interfaces

no implementation

can have public static fields, constants

all methods are public and abstract
no need to explicitly state