Format of Midterm

* 40 t/f
* 5 code evaluation
* 5 code understanding
* Writing code questions

Content

* Java syntax
* Expressions
* Value types vs. reference types
* Immutable objects
* For, while, if-else, switch
* Arrays, strings
* Classes, objects
* Enumerations(know what they are)
* Fields, methods
* Encapsulation
* Polymorphism
* Interfaces
* Inheritance

Overriding vs. overloading

|  |  |
| --- | --- |
| Overriding | overloading |
| Two identically named methods | Two identically named methods |
| One in parent class, one in child class | Both in same class |
| Same exact arguments | Different argument types/number of arguments |
| Overrides implementation | Provides different way to access method |

**Example of overriding**

Public class A{

Protected int I;

Public A(int i){

setI(i);

}

Public void setI(int i){

This. i = I;

}

Public int getI() {

Return I;

}

}

Public class B extends A{

Public B(int i){

Super(i)

}

@Override

Public void setI(int i){

If(i>0){

This.i = I;

}

}

}

***Example of overloading***

public class C{

protected int i;

public C(int i)[

this. i = i;

}

public void add(int a) {

return i+a;

}

public void add(int a, int b){

return i+a+b;

}

}

**Constructor chaining -** essentially a form of overloading, but with constructors rather than normal methhods

public class D{

protected int i;

public D(int i){

if(i>0){

this.i = i;

}

}

public D(){

this(10);

}

}

* Arraylists are not a valid built-in Java data type, you must import it
* reference types: you must use the word "new" to create an instance of this type

Variable names

* variables names can contain letters, digits, $, or \_
* digits cannot be the first thing in the name
* can't be a reserved keyword, such as while
* everything else is illegal

* block of statements can be empty
* object properties do not necesarily need a getter or a setter
* interfaces never specify constructors
* polymorphism allows methods to be context-specific
* encapsulation hides object state in order to support abstraction

int[] c = {0,1,2}

int []a = c;

a[0] = 10;

then c[0] = 10;

this happens because these are reference types, so c and a are pointing to the same array!