Midterm 1 Study Notes

Every java expression has its own type: `new A();` has a type of A. When we are declaring variables we generally declare both a static and dynamic type. For instance:

```
Object x = new A();
```

This object has a static type of Object and a dynamic type of A. However, this example isn really interesting because every object has a parent type of `Object`. Instead, we look at this example:

```
class A
class B extends A
class C extends A
class D

A s = new C();
```

Scope of Variables

Take this code:

```
class E{
    int f() ...
    int f(int x) ...
}

class F extends E{
    int f() ...
}
```

If you call f() as is without an argument then it would use the `f()` which is defined in `class F`. However, if we passed the function `f` with an integer argument then it would use the `f(int x)` function defined in `class E`. As an example:

```
E x = new F();
x.f() // this would call f() defined in class F
x.f(3) // this would call f() defined in class E
super().f() // this would call f() as as defined within class E
```

Miscellaneous things from before

Interfaces work just like classes, except when an interface is passed in as an argument you must call `a.function_name` on it to invoke the function.

For instance [Fall 2016 problem 4]:

```
    public interface BinaryPredicate {
        boolean test(int a, int b);
    }
```

This initializes an interfaced named BinaryPredicate, but when Java compiler runs it doesn't actually run it. However, there is already a test function built into it but the function has no arguments and does nothing yet.

When the problem statment has a header like this:

```
public static boolean allPairsCheck(BinaryPredicate p, IntList L)
```

Then it means that the interface p already has a `p.test()` function built into it that we can directly use. Another thing to note, we can run through IntLists like this:

```
for (IntList i = L, i.tail != null, i = i.tail)
```

Essentially this is the same statement as a regular for loop that goes through integers, but this one iterates over the entire linked list.

Interfaces

Interfaces are the same as classes except they are abstract - meaning that they cannot be called directly like `new InterfaceName()`, but instead we need to call the methods within the interface with `new MethodName()`.

They also need to be initialized, and usually we do that using the initialization of a new class:

```
class Nondescend implements BinaryPredicate {
    public boolean test (int x, int y){
    return x <= y
    }
}
```

Notice how the Nondescend class implements BinaryPredicate. This makes a new instance of BinaryPredicate, and within it we define the function `test(int x, int y)` to do what we want it to do. We usually do this with mapping functions or any type of function that iterates through something. In this case, it was used here:

```
public class Utils {
    public static boolean ascending(IntList L){
        return allPairsCheck(new Nondescend(), L)
    }
}
```

Here's another example that illustrates basically the same thing. Say we wanted to apply a function (like map) to all elements of an IntList. First what we need to do is define what kind of function we want. But since we want to keep it generalized, this necessarily implies that we cannot explicitly write down what procedure we wnat. Thus, we make an interface to handle that.

```
public interface IntUnaryFunction {
    int apply (int x);

    /* note that the function thus far has not been explicitly defined, we
just know that apply takes in a single argument and returns an integer
value.
    */
}
```

Now say we wanted to apply a specific function:

```
class square implements IntUnaryFunction{
    public int apply(int x){
        return x**2;
    }
}
```

This defines the square procedure. Now if we wanted to map that out onto a list, all we need to do is apply this to every value in the function.

```
public IntList map(IntUnaryFunction proc, IntList items){
    if (items.tail == null){
        return proc.apply(items.head)
    }
    else{
        return new IntList(proc.apply(items.head), map(proc, items.tail));
    }
}
```

Once we've defined these methods, we can write something like this:

```
public absolute implements IntUnaryFunction{
    int apply(x){
        return Math.abs(x);
    }
}

absolute = map(new absolute(), L)
```

Note: the above methods needed new defining becuase the methods within the parent method did not define how the method would run.

Pointer Diagrams

When doing reassignment problems: pointers point to whatever the old pointer was poiting to. For instance:

```


static Object[] M(Object left, Object right){
    return new Object[] {left, right};
}

static Object[] A(Object obj){
    return (Object []) obj // casting the value obj to an object[]
}
static void create() {
    Object [] L;

    L = M(null, M(null, M(null, nulll)));
    L = M(A(L)[1], L);
    A(A(L)[1])[0] = A(A(L)[0])[1];
    L = A(A(L)[0]);

    // First thing to note: don't be scared of the code! It really isn't
as bad as you might think it is.
}
```

Note here that `A((L)[0])` simply refers to the array object `L` and allows us to index into it. The second line, `M(A(L)[1], L)` means that the `right` object of `L` points to what `L` was pointing to before.

Some general things to keep in mind

Never be scared of what's on the midterm! The problems should be doable with the knowledge you have, just stay calm and you'll do fine on the exam!