Lab 0.2
Friday, Feb 12th  |  11AM-2PM  |  32-123
Read lab before coming (on Stellar)
Bring your laptop

Exploration 1
Due Monday, Feb 15th
Don't leave for last minute – requires more thought than labs
Brief Overview

- Interfaces
- Run time vs Compile time Type
- Overloading vs Overriding
- Exceptions
Interfaces

Define a type

Specifies a Contract
List the methods a class must implement
Only specifies methods – doesn't implement them

Substitution Principle
Anywhere interface type is asked for in code, we can use any class which implements that interface

Why?
Allows us to write generic code – write code which uses interface type, then can use any type of object which implements this interface
Runtime vs Compile time Types

Compile time type is type used in variable declaration.

Run time type is type of object variable refers to.
Called “Run time type” because may be unknown until program is actually executed.

Run time type of a variable may change as that variable is assigned new values.

```java
void placeAnimal(Animal animal) {
    // ...
}
```
Overloading vs Overriding

Overloading

*Same method name*

*Different parameter lists*

```java
void climb(Tree tree) {}
void climb(Cage cage) {}
```

Overriding

*Same method name*

*Same parameter lists*

*One method in subclass of other method's class*

```java
class Monkey {
    void climb(Tree tree) {}
}
}
class BabyMonkey extends Monkey {
    void climb(Tree tree) {}
}
```
Exceptions indicate an “exceptional” event – like an error in program execution

Exceptions thrown by a method are part of the method's specification

```java
public void transferTo( . . . ) throws InsufficientFunds {
    . . .
}
```

Indicates that any call to transferTo must be prepared to deal with problem of insufficient funds in account
Why not just use a return code to indicate error? (return null, or -1, or some “flag” value)

No guarantee it will be handled – caller might ignore return value and pass around null/-1/etc as if it were the actual answer

Different ways to fail – return code doesn't indicate which mode of failure occurred

Often help organize code into “standard case” and “error case”
Dealing with Exceptions

“Checked” exceptions MUST be either caught or propagated upward

Propagate exception upward to whoever called us:

```java
public Animal sellAnimal(...) throws InsufficientFunds {
    // get payment from buyer
    buyerAccount.transferTo(myAccount, 1000);

    return mother.breed();
}
```
Catching Exceptions

Put any code which might throw exception into try block

Add a “catch” block for each exception you'd like to catch

When exception thrown, execution jumps to first catch block which handles that particular exception

```java
try {
    dealer.sellAnimal(ZooVille.myAccount);
} catch (InsufficientFunds e) {
    System.out.println("You do not have suffic...");
}
```
Finally

Finally block is ALWAYS executed

Put any cleanup code into a finally block

```java
try {
    dealer.sellAnimal(ZooVille.myAccount);
}
catch (InsufficientFunds e) {
    System.out.println("You do not have suffic...");
}
finally {
    // cleanup
}
```
Homemade Exceptions

To create your own exception, just extend the Exception class

Exceptions are just like any other class – they can have fields, methods, etc

```java
public class AnimalUnavailableException extends Exception {
}
```

Then “throw” the exception

```java
public Animal sellAnimal(...) throws AnimalUnavailableException, InsufficientFunds {
    if(Math.random() < .15) {
        throw new AnimalUnavailableException();
    }
}
```
Whenever we use an object, we risk that object being null

dealer.buyMonkey(ZooVille.myAccount);

Throws a NullPointerException

Wildly inconvenient if these kinds of exceptions had to be handled – wrap every use of an object in a try catch block?

Instead rely on programmer to ensure that exception isn't thrown

If exception is unchecked, don't need to specify “throws” in method declaration
Which exceptions are unchecked exceptions?

Can just try calling method without try/catch, and see if code compiles

More systematic – all unchecked exceptions derive from either RuntimeException or Error