Review
A quick review of CS1
You should be familiar with most of this
There is some new material

package declarations (not used in CS2)
First program line of file
package declaration sets the package for all
declarations in the file
Not used in CS2
Example: "package RitIo;"

import declarations
Allows using shorter names to reference classes or
interfaces in other packages
Can either import a single type:
import java.util.Stack;
or all of the types in a package:
import java.util.*;

class declarations
Generally one class declaration per file:
modifiers class ClassName {
    class body declarations ...
}
class body declarations:
    fields:
        modifiers Type variables;
        modifiers Type variable [] = initializer;
    constructors:
        modifiers ClassName ( optional arguments ) {
            optional explicit constructor invocation;
            statements ... 
        }
    methods:
        modifiers returnType name ( optional arguments ) {
            statements ...
        }
    ...

modifiers
public, protected, private, (none)
    public - anyone can access
    protected - only the class and its subclasses can
                  access
    private - only the class itself can access
    (none) - only members of the package can access
abstract
class cannot be instantiated
method cannot be hidden or overridden
all interfaces are abstract so this would be redundant
static
not allowed for top-level classes
static methods and fields are associated with the class
and not the instance
static generally means "exactly one" rather than one per
instance
final
    a final class cannot be subclassed
    a final method cannot be hidden or overridden
    a final variable can only be assigned to once - generally
    used for constants
    (transient, volatile, synchronized, native, strictfp)
don’t worry about these for now (or ever for some of
these)

Statements
local variable declaration
    int i = 3;
block
    ( statements )
empty
    ;
return
    return;
    return expression;
if then
    if ( expression ) {
        statements ...
    }
if then else
    if ( expression ) {
        statements ...
    } else {
        statements ...
    }
switch
    switch ( expression ) {
        case: statement;
        break;
        default: statement;
    }

Loop statements
labeled
    label: statement;
break
    break;
    break label;
continue
    continue;
    continue label;
while
    while ( expression ) {
        statements ...
    }
for
    for ( initialization ; test ; update ) {
        statements ...
    }
do
    do {
        statements ...
    } while ( expression )
Exception handling statements

```java
throw
  throw expression;

try
  try {
    statements ...
  } catch ( exceptionClass e1 ) {
    statements ...
  } catch ( exceptionClass e2 ) {
    statements ...
  } finally {
    statements ...
  }

(compatible) synchronized

expressions

assignment:
  variable = expression;

increment / decrement:
  variable ++;
  variable --;
  ++ variable;
  -- variable;

method and constructor invocation:
  expression . name( args );
  new className( args );

Operators:

?:
  ||
  & &
  | |
  ^
  @
  == and !=
  < > <= >=
  << >> >>>
  + -
  * / %
  unary + -
  unary - ~
  postfix ++ --

cast
  ( type ) expression

literal
  this
  ( expression )
  new className( arguments )
  x . i
  Class . i
  x . f( arguments )
  Class . f( arguments )
  x[ i ]
```
Inheritance - Superclasses and subclasses

A subclass is said to “extend” the superclass
A subclass inherits methods and fields from the superclass
All methods of the superclass are available to the subclass
A subclass can redefine a method - this is called “overriding the method”
Syntax for declaring a subclass

class Subclass extends Superclass { }

is-a relationship

The superclass - subclass relationship implements the “is-a” relationship between different classes of objects
a subclass must have all of the state and behavior of its superclass and maybe more
a square is a rectangle
a circle is a shape
a rectangle is a shape
a square is a rectangle
All subclasses of a superclass must have at least the same state and behavior as the superclass

Variables and Subclasses

A variable declared to refer to a class can refer to any subclass of that class
You can only call methods defined for the variable’s class

Accessing subclass methods and fields

When an object’s method is referenced, it is the object’s class that determines which method is referenced, not the class of the variable that references it
Method invocations always use the true object class for finding the method to run - not the class of the variable
When an object’s field is referenced, it is the variable’s class that determines which field is referenced, not the class of the object that it references

Access modifiers

public
Anyone can access
protected
Only the class and its subclasses can access
private
Only the class can access

(no)
Only members of the same package can access

Constructor Declarations

First statement is an optional explicit constructor invocation
can be either a call to another constructor in the same class

this( optional arguments );

or a call to a constructor in the direct superclass

super( optional arguments );

If not present then an implicit call to super(); is assumed

The final modifier

final classes cannot be subclassed
final methods cannot be overridden
final fields must be assigned to exactly once
static final fields should be initialized in the declaration
static instance fields must be assigned at the end of every constructor

Abstract classes

sometimes the superclass is a superclass to several subclasses and it does not make sense to make objects of the superclass class
A Shape class may have several geometrical shapes as subclasses but a Shape object that is not a specific shape may not make sense
Abstract classes are classes that define signatures for methods and declare fields but objects of the class cannot be instantiated
An abstract class can declare abstract methods
An abstract method has the modifier “abstract” in the declaration and a semicolon instead of a body:

public abstract double area();

An abstract class defines the behavior of all subclasses although it may not actually implement any methods to define this behavior
Every subclass of an abstract class must either be abstract or override all methods that were declared abstract