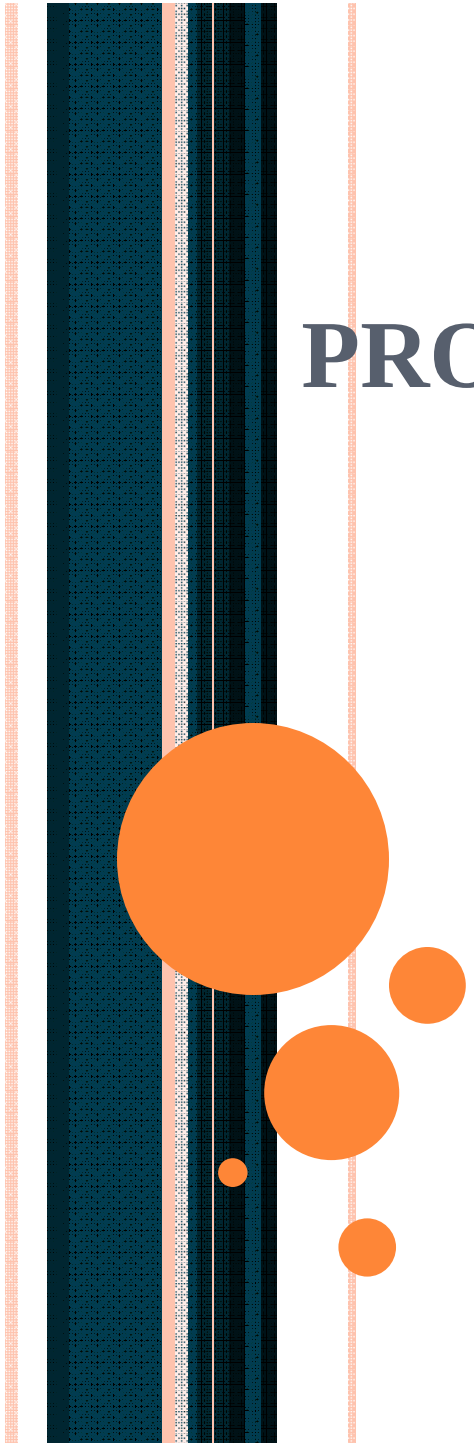


OBJECT ORIENTED PROGRAMMING USING C++

Access Modifiers

- Control which classes use a feature
- Only class-level variables may be controlled by access modifiers
- Modifiers
  1. public
  2. protected
  3. private
- Non-inner classes can only be public

Friendly

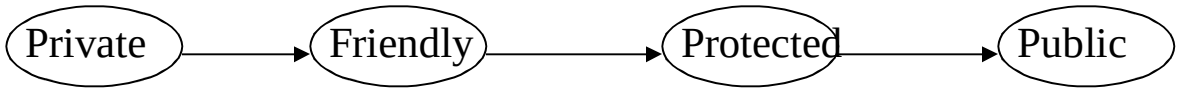
- Features with no access modifier default to friendly
- Friendly features are accessible to any class in the same package
- Classes outside the package may not access these features
- Friendly classes may be subclassed but their variables/methods are not accessible by the subclass

Modifiers

- public
  - Can be used in any Java program without restriction
- private
  - may only be used by the instance of the class that declares the variable or method
- protected
  - only variables, methods, and inner classes can be declared protected
  - available to all classes in the same package
  - available to all subclasses(even those in different packages)

Overriding Methods

- Methods may not be overwritten to be more private



Final Modifier

- Final features may not be overwritten
- A final class may not be subclassed
- A final variable cannot be changed once it has been assigned a value

Final Modifier Example

```
class Walrus {
  int weight;
  Walrus( int w ) { weight = w };
}

class Tester {
  final Walrus w1 = new Walrus(1500);
  void test() {
    w1 = new Walrus(1400); // Illegal
    w1.weight = 1800;      // Legal
  }
}
```

Abstract Modifier

- An abstract class cannot be instantiated
- This is a way to defer implementation to subclasses
- An class with one more methods declared abstract cannot be instantiated
- A class that is declared to implement an interface but does not implement all the methods of that interface must be abstract
- Similar to virtual in C++

Abstract Example

```
abstract class Stack {
  protected int count = 0;

  public abstract void push( Object o );
  public abstract void pop();
  public abstract Object top();
  public abstract boolean isFull();

  public boolean isEmpty() {
    return count==0;
  }
}
```

Static Modifier

- Associated with a class, not a particular instance of a class

```
public class StaticTest {
    static int x = 0;
    StaticTest() {
        x++;
    }
}
```

No matter how many instances of StaticTest we have the ‘x’ variable is the same for all

Accessing Static Variables

```
StaticTest st = new StaticTest();
st.x = 69;
```

OR

```
StaticTest.x = 69
```

More about Static

- Static methods cannot use non-static features of their class
- They can access the class's static data
- Since static methods are not associated with an instance of a class there is no *this* variable

Static Initializers

```
public class Demo {
    int x = 5;

    static {
        x = 69;
    }

    public static void main( String[] ) {
        System.out.println( ‘X = ‘ + x );
    }
}
```

What is the value printed?

Native modifier

- Can only refer to methods
- Indicates that the body of the method is to be found elsewhere
- Namely in a file in another language
- Call to native method is the same as if it was implemented in Java

Transient modifier

- Only applies to variables
- Transient variables will not be serialized
- Transient variables cannot be final or static

Synchronized modifier

- Used to control critical code in multi-threaded programs
- Covered in chapter 7

Volatile modifier

- Not on exam
- Used in multiprocessor environments
- Applies only to variables