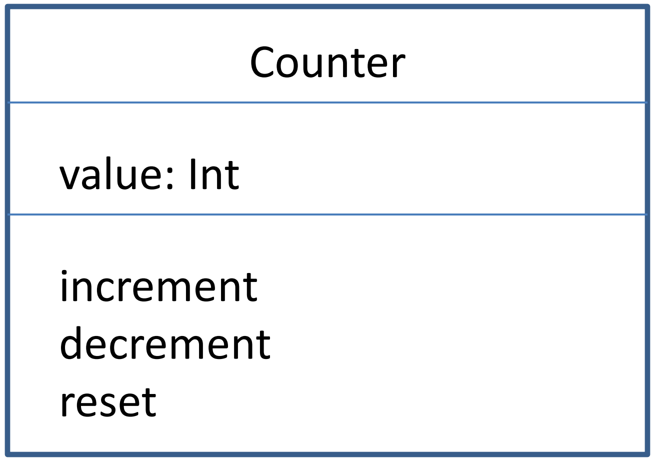
Practice

Lecture 4

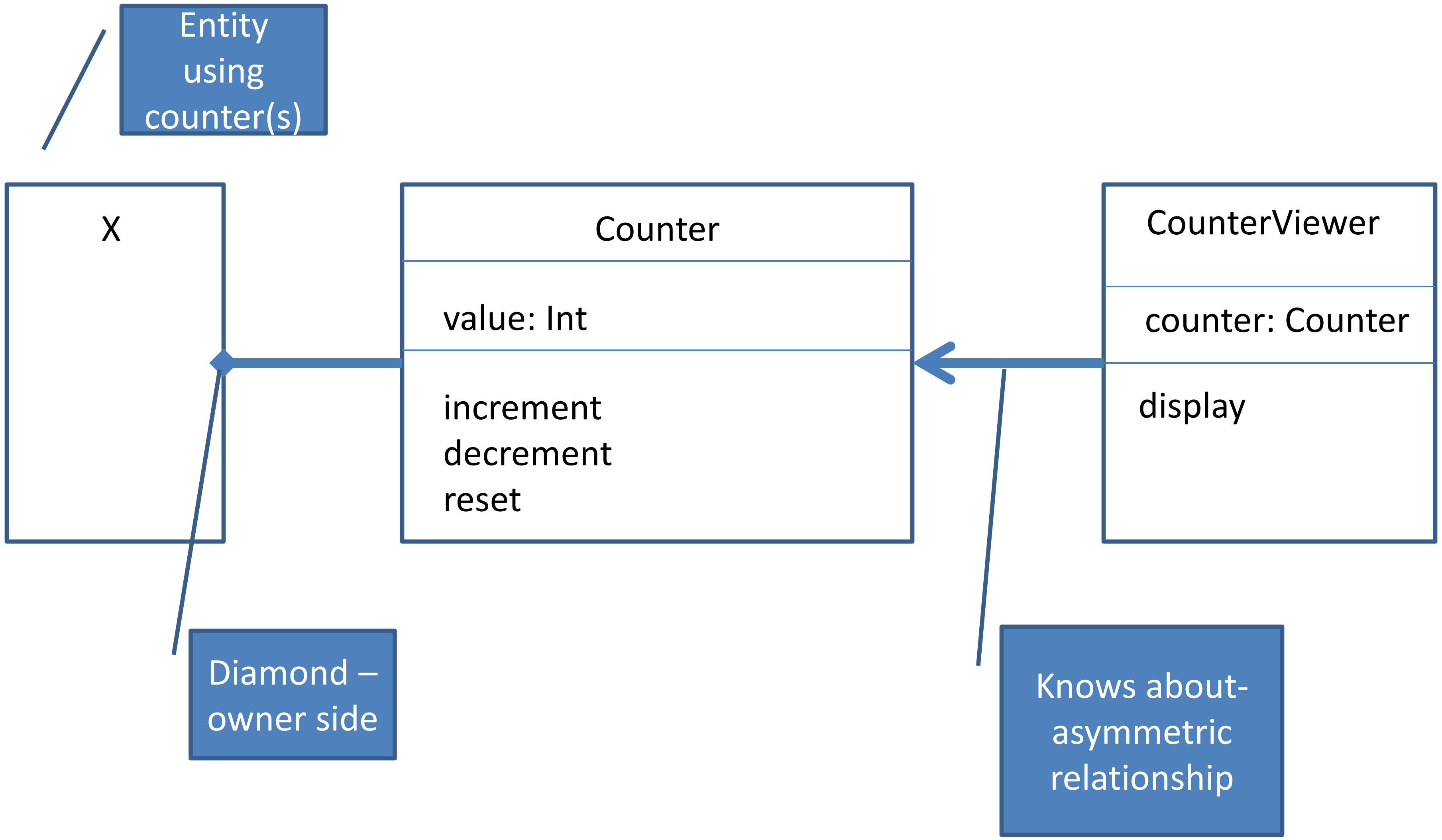
The model and the view

- Model: includes objects
- View: defines an interaction with external world (not included into the model)
  - User interface
  - Data management system

Counter class: model



System with *Counter*



I always keep my variables private. If you want to see them, you have to talk to my methods.
private

Declarations: class *Counter*

```
Public class Counter()
{
            private int   _value;
            public Counter()
            {
            }
            public void increment()
            {
            }
            public void decrement()
            {
            }
            public void reset()
            {
            }
}
            public int getValue()
            {
            }
            public void setValue(int value)
            {
            }
}
```

Declarations: class *Counter*

```
public class Counter()
{
            private int   _value;
            public Counter()
            {
            }
            public Counter(int initialValue)
            {
            }

            public void increment()
            {
            }
            public void decrement()
            {
            }
            public void reset()
            {
            }
}
```

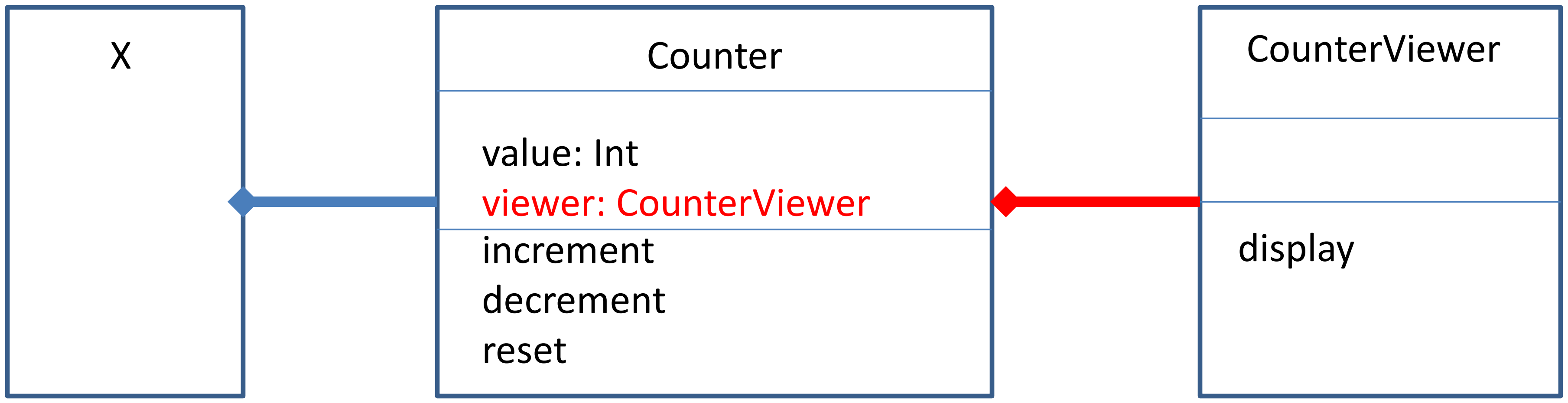
Declarations: class *CounterTextGUI*

```
public class CounterTextViewer()
{
        private Counter _counter;
        public CounterTextViewer(Counter counter)
        {
                _counter=counter;
        }
        public void display()
        {
        }
}
```

Class *CounterTest*

```
public class CounterTest()
{
        public CounterTest(Counter counter, CounterTextGUI viewer)
        {
        }
        public static void main(String [] args)
        {
                Counter myCounter=new Counter();
                CounterTextViewer counterViewer=new CounterTextViewer (myCounter);
                 CounterTest test=new CounterTest(myCounter,counterViewer);
                //what to test
                test.testInitialState();
                test.testIncrement();
                test.testDecrement();
                test.testReset();
        }
}
```

Alternative system with *Counter*

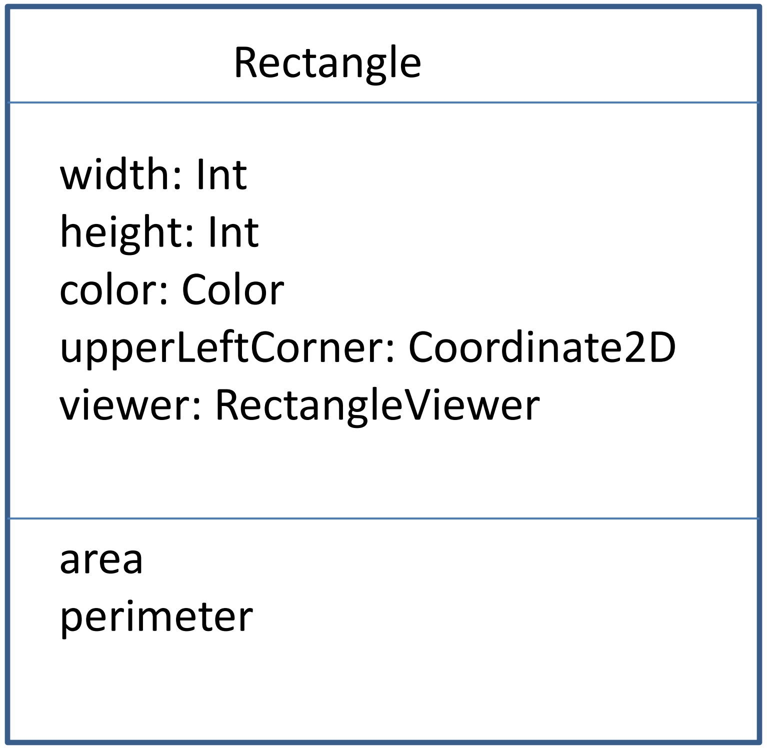


How would the constructor for the class Counter change?

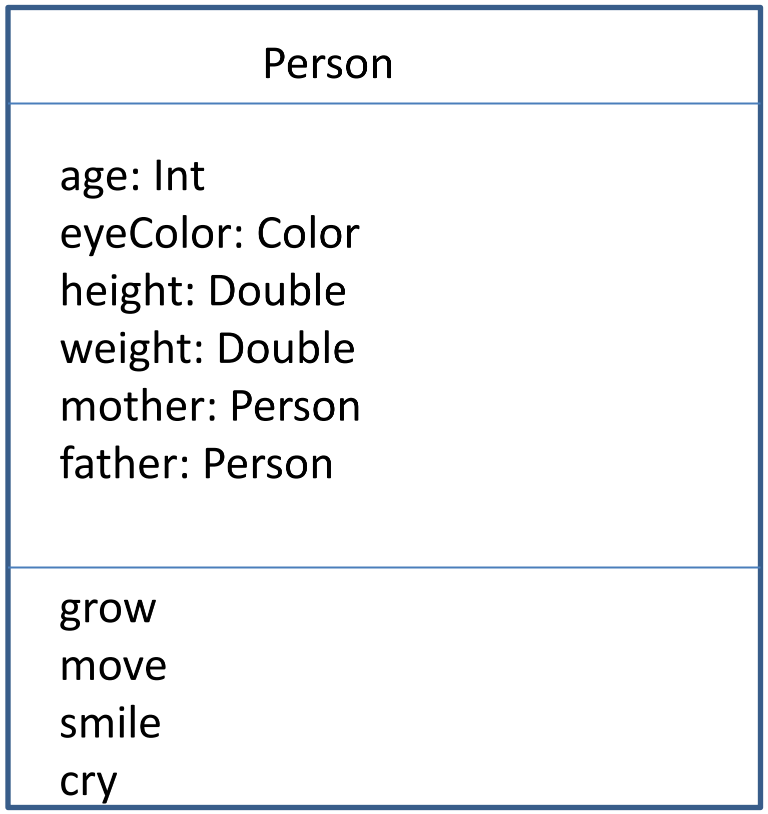
Designing classes

- Rectangle
- ElectricBulb
- Person

Class *Rectangle*



Class *Person*



What to test

- Initial state
- State after each command
- Return values from each query

Designing systems

- Nim game

Java basics

1. Console input and output
2. System object
3. Primitives and primitive wrappers
4. From String to numeric conversions
5. Formatting numbers
6. Compiling and running
7. Program arguments
8. Packages
9. Access specifiers: default, public, private
10. Static variables and methods

Console output

```
System.out.print(“Your name is “+name);
System.out.println(“You have “+counter+” cards left”);
```

Cross-platform: System object

```
System.getProperties().list(System.out);
System.out.println(System.getProperty("user.name"));
System.out.println(   System.getProperty("java.library.path") );
```

Important for now:

```
System.getProperty(“line.separator")
System.getProperty(“file.separator")
```

Path to the file

```
String fullFileName =
        dir+System.getProperty(“file.separator”)+fileName;
```

Multi-line output

You can:

```
System.out.print (“Name: “+name+”\n”+”Age: “+age);
```

But to be platform-independent always do:

```
System.out.print (“Name: “+name+System.getProperty(“line.separator")
                                                   +”Age: “+age);
```

Console input: reading lines

```
BufferedReader reader = new BufferedReader(new
                                InputStreamReader(System.in));


System.out.print("Please enter user name : ");
String username = null;
try {
        username = reader.readLine();
} catch (IOException e) {
      e.printStackTrace();
}
System.out.println("You entered : " + username);
```

Java primitives

Integral types:

Floating-point:

Boolean:

String:

Java primitives

Integral types: *byte, short, int, long, char*(2 bytes)

Floating-point: *float, double*

Boolean: *boolean*

int i=25;

long k=i;

int m=(int) k;

Explicit conversion is required

String – non-primitive data type

String s="abcd"; // is a shortcut to

String s=new String("abcd");

String s=new String(); //puts an empty string ""

Wrapper classes

byte	*Byte*
short	*Short*
int	*Integer*
long	*Long*
char	*Character*
float	*Float*
double	*Double*
boolean	*Boolean*

Wrapping and unwrapping

```
int i=23;
Integer iWrap=new Integer(i);

int j=iWrap.intValue();
```

How to convert from String to number

Reader.readLine returns *String*

What if we need to get a numeric input from a user?

Useful methods of Wrapper objects: type conversion

```
String s=new String("124");
int s=Integer.parseInt(s);



int i=123;
String iString=String.valueOf(i);
```

Number formatting

- Formatting is not only part of the I/O system, but also of a String object
- *Formatter* class

```
String s = String.format(“%, d”, 1000000000);
System.out.println(s);
```

```
1,000,000,000
```

Number formatting examples

String s = **String.format("%, d", 1000000000);**

$\uparrow$

The percent (%) says, "insert argument here"

format("I have %.2f bugs to fix.", 476578.09876);

```
I have 476578.10 bugs to fix
```

format("I have %,.2f bugs to fix.", 476578.09876);

```
I have 476,578.10 bugs to fix
```

Steps in developing a Java application

1. Draw UML diagram of all classes needed
2. Check what classes you have available
3. Declare all new classes: variables and methods. Compile
4. Write tests for each class
5. Implement. Compile
6. Run tests for individual classes
7. Test the entire system

Starting point of your program - *main*

The code is in main only for two things:

- to test your class
- to start the application

Starting point of your program - *main*

```
public static void main(String[] args) {
    MiniMiniMusicCmdLine mini = new MiniMusicCmdLine();
    if (args.length < 2) {
      System.out.println("Don't forget the instrument and note args");
    } else {
      int instrument = Integer.parseInt(args[0]);
      int note = Integer.parseInt(args[1]);
      mini.play(instrument, note);
    }
  }
```

Compiling and running your code

```
C:> dir greetings/
C:> dir greetings Hello.java
C:> cat greetings\Hello.java
    package greetings;
    public class Hello
    {
        public static void main(String[] args)
        {
                for (int i=0; i < args.length; i++)
                        System.out.println("Hello " + args[i]);
        }
    }
C:> javac greetings\Hello.java
C:> dir greetings
Hello.class Hello.java
C:> java greetings.Hello World Universe Everyone
Hello World
Hello Universe
Hello Everyone
```

Name visibility. Name clashes

- If two programmers are working on the same system, they may give identical names to their variables and methods
- In C++ use *namespace*
- Still, global variables are permitted, and this causes name collisions

Name visibility. Java approach

$\underbrace{\text{com.barskym}}_{\text{Reverse domain name}}\text{.oopdemos.MiniMusicPlayer}$

All of your files automatically live in their own namespaces, and each class within a file must have a unique identifier—the language prevents name clashes for you

Fully qualified object name

```
package com.barsky.oopdemos;
```

To use:

```
com.barskym.oopdemos.MiniMusicPlayer player
        =new com.barskym.oopdemos.MiniMusicPlayer ();
```

Using libraries

import java.util.ArrayList; - import class

import java.util.*; //import package (library of classes)

ArrayList lst=new ArrayList();

Package caveat

- Anytime you create a package, you implicitly specify a directory structure when you give the package a name.
- The package *must* live in the directory indicated by its name
- This must be a directory that is searchable starting from the CLASSPATH.
- Note that compiled code is often placed in a different directory than source code, but the <u>*path to the compiled code*</u> must still be found by the JVM using the CLASSPATH.

Setting CLASSPATH

```
C:> java -classpath C:\java\MyClasses
                        utility.myapp.Cool
```

```
C:> java -classpath
      C:\java\MyClasses;C:\java\OtherClasses
                              utility.myapp.Cool
```

Arguments after **java** are JVM arguments,

arguments after the program name are program arguments

Access specifiers

If no specifier – package access

public

private

protected - later

The static keyword

- Without new – no object, no fields, no methods
- If you want to have only a single piece of storage for a particular field (pi), regardless of how many objects of that class are created, or even if no objects are created.
- If you need a method that isn’t associated with any particular object of this class (random(), max()).
- Then use **static** keyword.

Static field

```
class StaticTest {
        static int i = 47;
}

StaticTest.i++;
```

Static method

```
class Incrementable {
        static void increment() { StaticTest.i++; }
}
```

Note: You cannot access non-static fields and methods inside the static method

```
Incrementable sf = new Incrementable();
sf.increment();
```

or

```
Incrementable.increment();
```

Code conventions

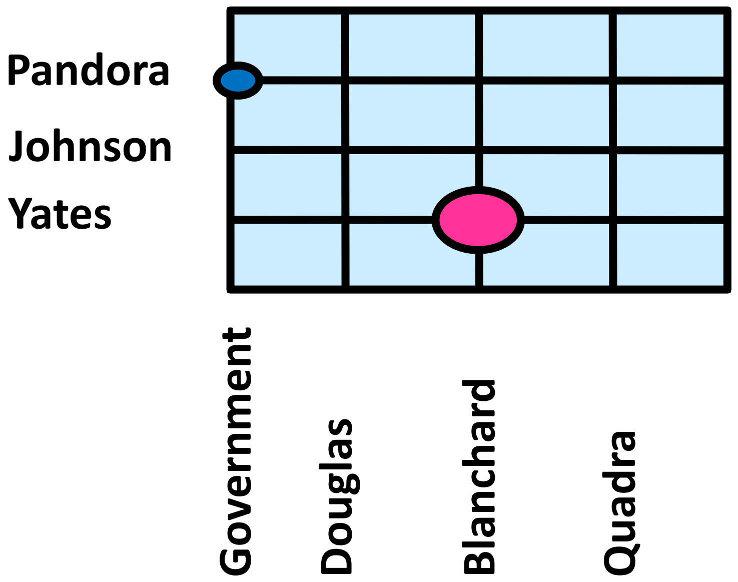
- http://java.sun.com/docs/codeconv/html/CodeConventions.doc.html

- Textbook 1.9, 1.10

Coding assignment 1

- Design and implement an application which moves a vehicle from the current intersection to the destination intersection and back in a mini-city of Victoria. Vehicle has the ability to turn and to move forward, one block at a time.
- Write tests for every class.
- Create text-based user interface which would supply an inexperienced driver with exact instructions

Map example



User input example:

$$\underbrace{\text{“Government”,”Pandora”,}}_{\text{fromIntersection}}\ \underbrace{\text{”Yates”,”Blanchard”}}_{\text{toIntersection}}$$

Java arrays

```
String [] streetNamesX = {“Pandora”,”Johnson”,”Yates”};

String [] streetNamesY =
        {“Government”,”Douglas”,”Blanchard”,”Quadra”};
```

These arrays can be hardcoded into your application

Assignment 1 info

- 7 points
- Due: September 19, before the start of the next lab