



EECS 1720

Building Interactive Systems

Lecture 7/8 :: Encapsulation

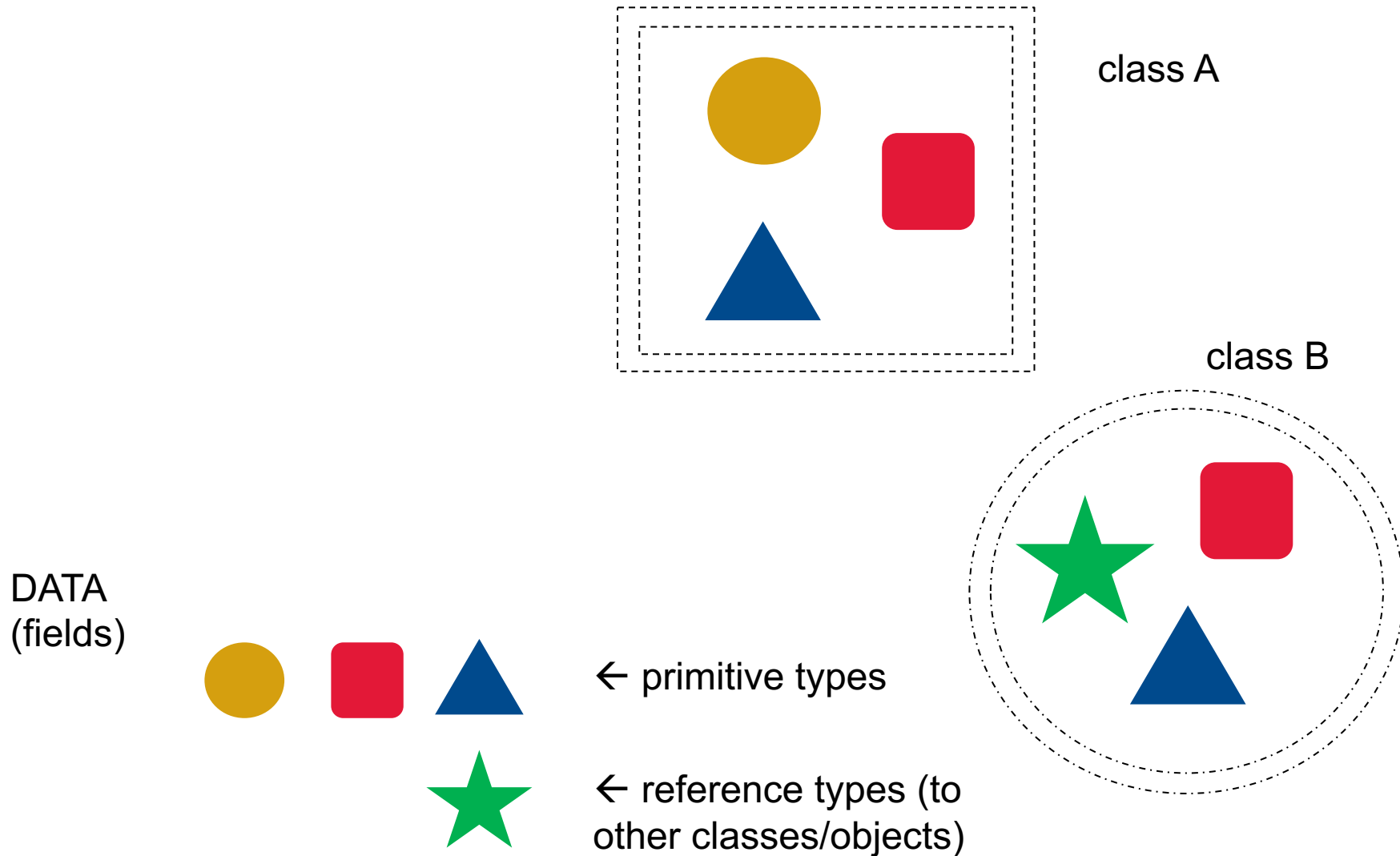
Topics:

- Encapsulation
 - Access (public vs. private)
 - Constructors (default, custom & copy)
 - Methods (accessors/getters, mutators/setters)
- Class Relationships
 - Association (has-a) vs. Inheritance (is-a) ← high level
 - Other options
 - Multiple classes in a file?
 - Classes within classes
- Graphics2D introduction (live session)

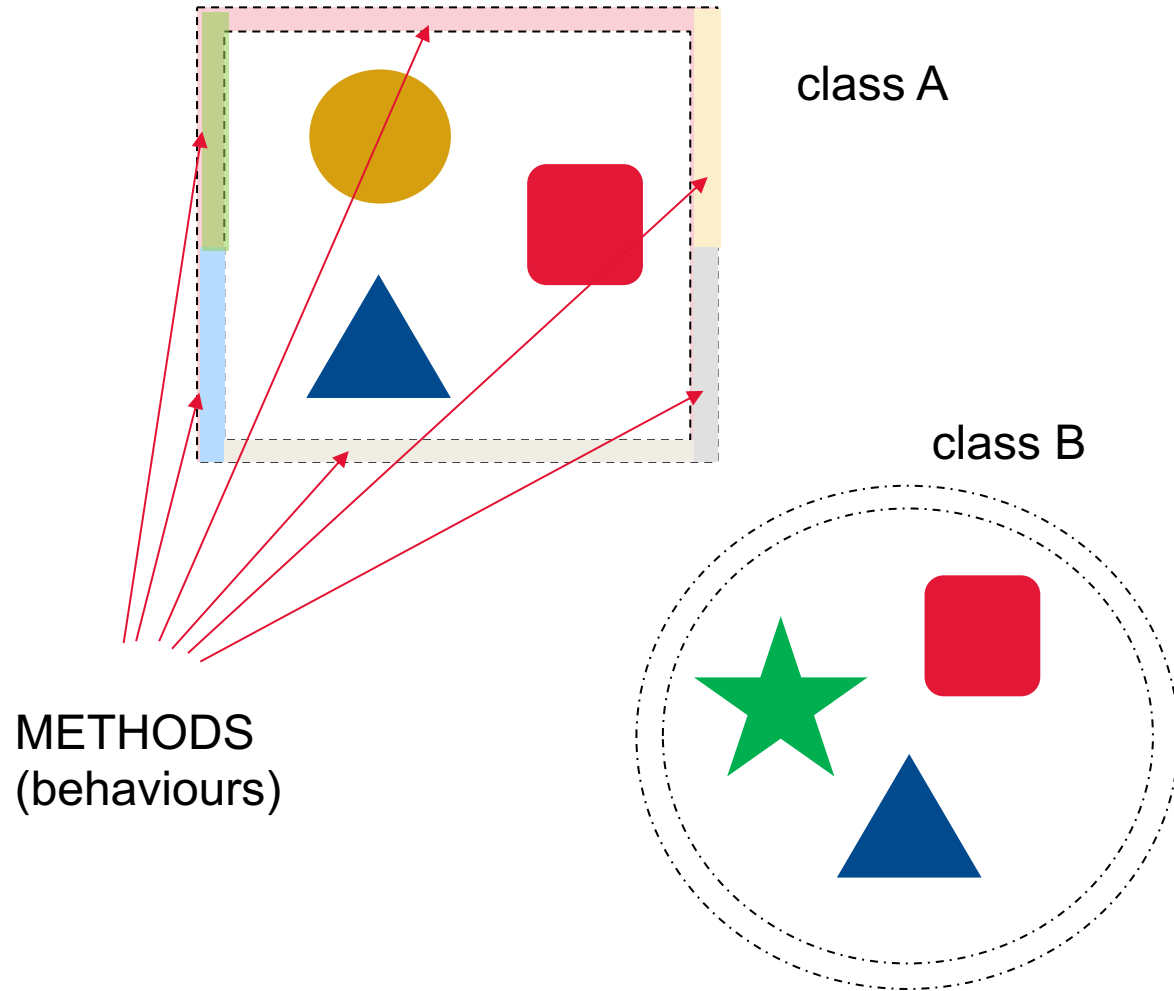
Recall: Exception objects

- E.g. RuntimeException
 - Like any object, we can instantiate a RuntimeException object
 - Exception objects include:
 - Data
 - Information about the error/issue (including its type/message)
 - State of the program when the error/issue occurred (stack trace)
 - Behaviour
 - “throwable” – making them compatible with Java’s throw mechanism – a way to transfer control from one part of the program to another to circumvent/recover or gracefully exit when an unexpected error occurs during runtime
 - We say that Exception objects “**encapsulate**” the notion of an exception
 - Similarly, other classes “encapsulate” other notions/things
 - What do we mean by “encapsulate”??

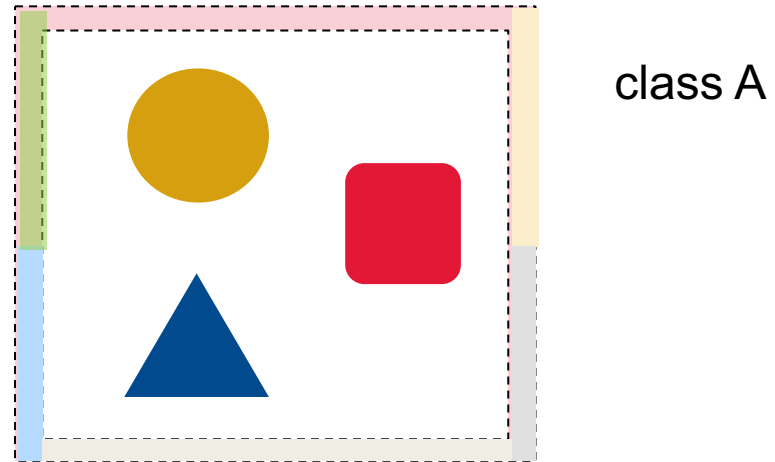
Classes combine data/state



Classes combine function/behaviour



ENCAPSULATION



ENCAPSULATION =

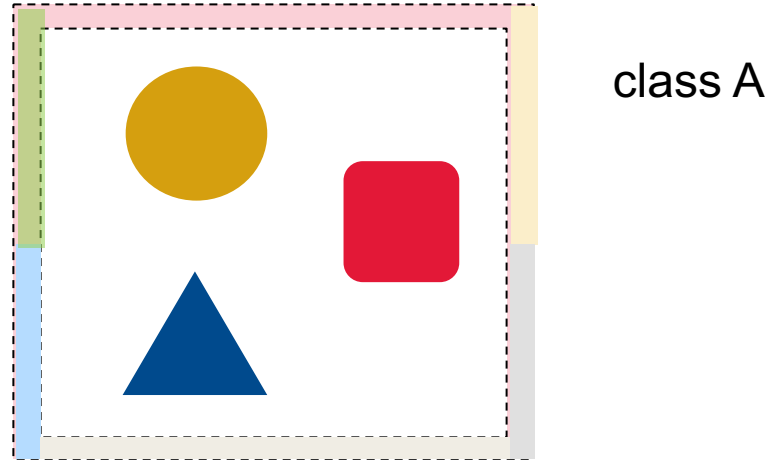
How data & methods are “combined together” & managed within a given shell (i.e. a class)

Specifically,

Encapsulation relates to

- how information/data is secured &
- how we manage interaction with that information

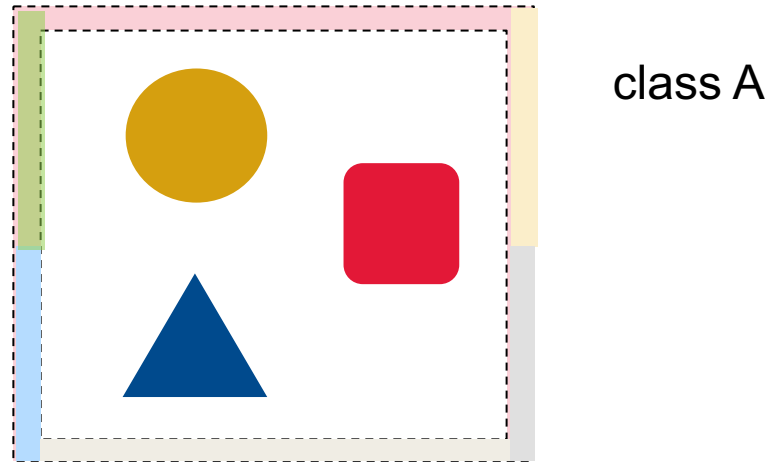
Classes ENCAPSULATE functionality



Why Encapsulate??

1. **ORGANIZATION**: Co-location of related data and methods (e.g. Math Class co-locates PI, etc., with common math methods like pow(), sqrt(), etc.
2. **PORTABILITY**: ease of transport – when classes used to make **objects**, each individual object ***"carries around"*** its own version of the data (state) and the set of methods that are allowed to operate on its data

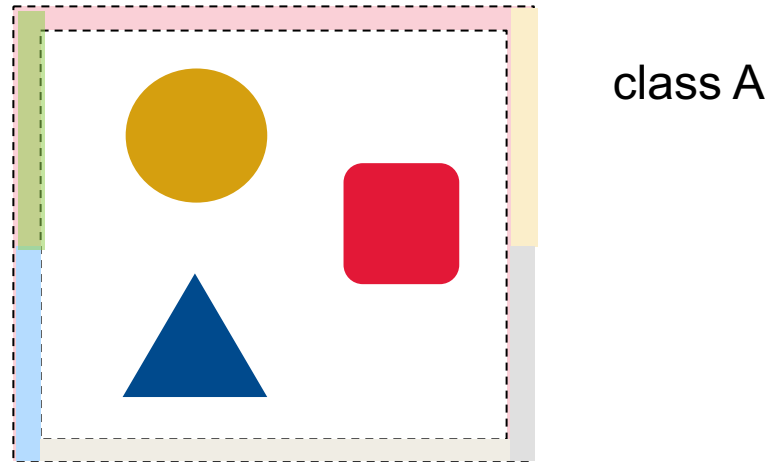
Classes ENCAPSULATE functionality



2. PORTABILITY:

these methods can have exclusive access to the data carried around within an object (without having to pass parameters)

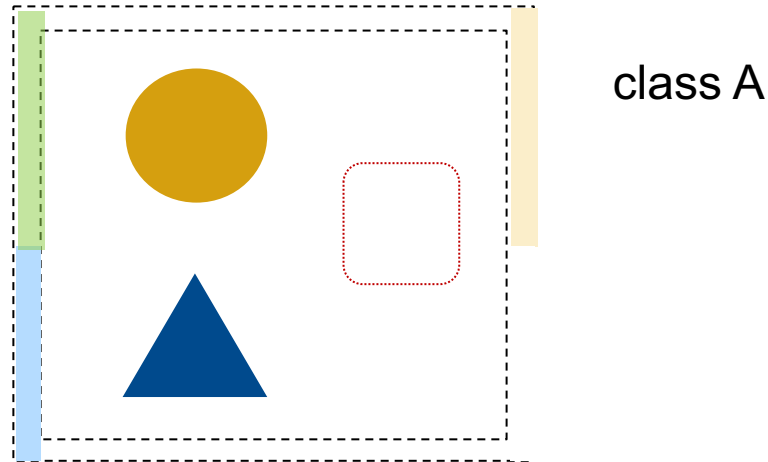
Classes ENCAPSULATE functionality



3. **ACCESS**: we can **control access** to parts of a class/object

- Internal: class methods have **exclusive access** to the data carried around within an object (without having to pass parameters)
- External: **we can restrict client access** to some features

Classes ENCAPSULATE functionality



3. **ACCESS**: we can control access to parts of a class/object

- Internal: class methods have **exclusive access** to the data carried around within an object (without having to pass parameters)
- External: **we can restrict client access** to some features

Class internals (closer look)

ACCESS

Fields

- Recall: fields are class variables inside the class, but external to any methods
- Access modifiers
 - **public**
 - field is accessible to any clients of the class
 - **private**
 - field is accessible only to methods/constructors inside the class
 - **protected**
 - we will look at this later
 - No modifier specified?
 - Acts as private, but accessible as public to any class in the same package (“package private”)

Methods

- Also have access modifiers!
 - **public**
 - **private**
 - **protected** (later)
 - no modifier

What do we mean by “public” or “private” ?

- Assume we have 2 classes (A and ClientOfA):

A.java

```
package week4;

public class A {

    public static int field1 = 5;

    public static int method1(int i) {
        field1 = field1 + i;
        return field1;
    }

    public static void main(String[] args) {
        // implementation not shown
    }
}
```

ClientOfA.java

```
package anotherPackage;

import week4.A;    // not needed (if in same package)

public class ClientOfA {

    public static void main(String[] args) {

        int localVar = 5;

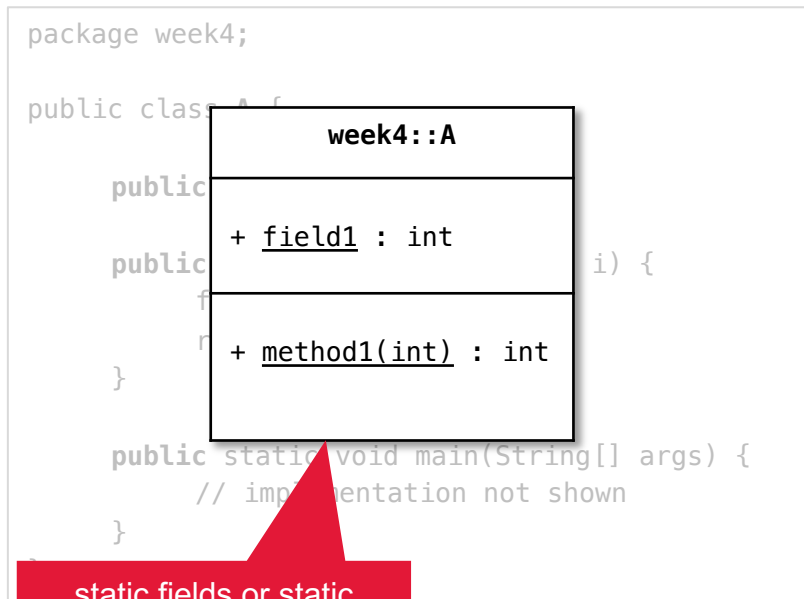
        System.out.println(localVar);

    }
}
```

What do we mean by “public” or “private” ?

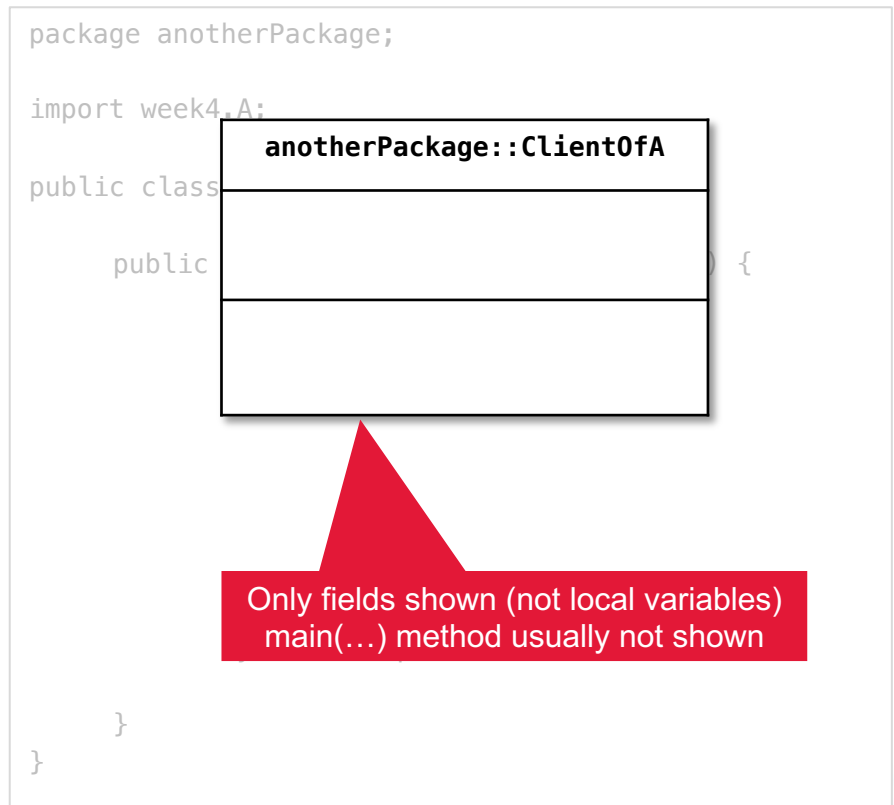
- UML?

A.java



static fields or static methods are underlined

ClientOfA.java



Only fields shown (not local variables)
main(...) method usually not shown

What do we mean by “public” or “private” ?

A.java

```
package week4;

public class A {

    public static int field1 = 5;

    public static int method1(int i) {
        field1 = field1 + i;
        return field1;
    }

    public static void main(String[] args) {
        // implementation not shown
    }
}
```

ClientOfA.java

```
package anotherPackage;

import week4.A;

public class ClientOfA {

    public static void main(String[] args) {

        int localVar = 5;

        System.out.println(localVar);

    }
}
```


What does “public” mean ?

- Assume we have 2 classes (A and ClientOfA):

A.java

```
package week4;

public class A {

    public static int field1 = 5;

    public static int method1(int i) {
        field1 = field1 + i;
        return field1;
    }

    public static void main(String[] args) {
        // implementation not shown
    }
}
```

ClientOfA.java

```
package anotherPackage;

import week4.A;

public class ClientOfA {

    public static void main(String[] args) {

        int localVar = 5;

        localVar = A.field1;    // ok
        A.field1 = 6;           // ok

        localVar = A.method1(localVar); // ok

        System.out.println(localVar);

    }
}
```

What does “private” mean ?

- Assume we have 2 classes (A and ClientOfA):

A.java

```
package week4;

public class A {

    private static int field1 = 5;

    public static int method1(int i) {
        field1 = field1 + i;
        return field1;
    }

    public static void main(String[] args) {
        // implementation not shown
    }
}
```

ClientOfA.java

```
package anotherPackage;

import week4.A;

public class ClientOfA {

    public static void main(String[] args) {

        int localVar = 5;

        localVar = A.field1;      // NOT ok
        A.field1 = 6;             // NOT ok

        localVar = A.method1(localVar); // ok

        System.out.println(localVar);

    }
}
```

COMPILE ERROR (client
does not have access to
A's private fields!!)

What does “private” mean ?

- Assume we have 2 classes (A and ClientOfA):

A.java

```
package week4;

public class A {

    private static int field1 = 5;

    private static int method1(int i) {
        field1 = field1 + i;
        return field1;
    }

    public static void main(String[] args) {
        // implementation not shown
    }
}
```

ClientOfA.java

```
package anotherPackage;

import week4.A;

public class ClientOfA {

    public static void main(String[] args) {

        int localVar = 5;

        localVar = A.field1;      // NOT ok
        A.field1 = 6;             // NOT ok

        localVar = A.method1(localVar); //NOT ok

        System.out.println(localVar);

    }
}
```

COMPILE ERROR (client
does not have access to
A's private methods!!)

So why have this?

- Designer of class A controls (restricts) what client can do with A
 - private features are locked from unauthorized use
 - they are only directly accessible to methods within class A
 - facilitates “*information hiding*” & “*improved integrity/security*”
 - Client cannot see how class A implements its functionality
 - Client can only modify fields in a controlled way (determined only by public methods that allow for it – if they are defined)
 - in previous example:
 - `method1` is public and controls how client can modify `field1`
 - `method1` only allows the client to increment `field1` by the argument `i`
 - If client has direct access, they can modify `field1` any way they like!

So why have this?

- In previous example:

```
package week4;

public class A {

    private static int field1 = 5;

    public static int method1(int i) {
        field1 = field1 + i;
        return field1;
    }

    public static void main(String[] args) {
        // implementation not shown
    }
}
```

method1 is public and controls how client can modify **field1**

method1 only allows the client to increment **field1** by the argument **i**

If client has direct access, they can modify **field1** any way they like!

Information Hiding?

```
package week4;

public class A {

    public static int field1 = 5;

    public static int method1(int i) {
        field1 = field1 + i;
        return field1;
    }

    public static void main(String[] args) {
        // implementation not shown
    }
}
```

```
package week4;

public class A {

    private static int field1 = 5;

    public static int method1(int i) {
        field1 = field1 + i;
        return field1;
    }

    public static void main(String[] args) {
        // implementation not shown
    }
}
```

week4::A
+ <u>field1</u> : int
+ <u>method1(int)</u> : int

private features (fields,
methods or constructors)
do not show in the public API

API = only public features

week4::A
- <u>field1</u> : int
+ <u>method1(int)</u> : int

package“private”:

A.java

```
package week4;

public class A {

    // no access modifier ?
    static int field1 = 5;

    private static int method1(int i) {
        field1 = field1 + i;
        return field1;
    }

    public static void main(String[] args) {
        // implementation not shown
    }
}
```

COMPILE ERROR (client
does not have access to
A's private methods!!)

ClientOfA.java

```
package anotherPackage;

import week4.A;

public class ClientOfA {

    public static void main(String[] args) {

        int localVar = 5;

        localVar = A.field1;    // NOT ok
        A.field1 = 6;           // NOT ok

        localVar = A.method1(localVar); //NOT ok

        System.out.println(localVar);

    }
}
```

package“private”:

- ClientOfAToo.java (class within same package as A)

A.java

```
package week4;

public class A {

    // no access modifier ?
    static int field1 = 5;

    private static int method1(int i) {
        field1 = field1 + i;
        return field1;
    }

    public static void main(String[] args) {
        // implementation not shown
    }
}
```

ClientOfAToo.java

```
package week4;

public class ClientOfAToo {

    public static void main(String[] args) {

        int localVar = 5;

        localVar = A.field1;    // ok
        A.field1 = 6;           // ok

        localVar = A.method1(localVar); //NOT ok

        System.out.println(localVar);

    }
}
```

considered public (by default) to all classes in same package

Class internals

Constructors/Methods

RasterImage class

imagePackage

Class RasterImage

java.lang.Object
imagePackage.RasterImage

All Implemented Interfaces:

java.lang.Iterable<Pixel>

```
public class RasterImage
extends java.lang.Object
implements java.lang.Iterable<Pixel>
```

This class encapsulates a raster graphics image, which is an image that is composed of a rectangular grid of pixels. The raster graphics image is displayable at runtime through the use of the `show` method. The window will have the default window decorations and the string that appears as the title is an attribute of the RasterImage object. This class provides constructors for the instantiation of a raster graphic image (i) from a file name, (ii) from another image, or (iii) from a width and height specification. This class provides accessor methods for the pixels of the raster image.

imagePackage :: RasterImage

// no visible (public) fields

// constructors

```
+ RasterImage()
+ RasterImage(int, int)
+ RasterImage(RasterImage)
+ RasterImage(String)
```

// other methods (not all shown)

```
+ getGraphics2D() : Graphics2D
+ getHeight() : int
+ getWidth() : int
+ getPixel(int,int) : int
+ getPixelColor(int, int) : int
+ setTitle(String) : void
+ setBasicPixel(int, int, int) : void
+ show() : void
+ toString() : String
```

Very similar to PImage class (in processing)
(however can show an image in its own
stand-alone window vs. a processing app)

Recall: Constructors & Methods

- Signatures are used by JVM to disambiguate between which constructor/method to use
- Constructors all have same name as the class
- Constructors can be declared multiple times (overloaded)
 - What makes them different is their signature
- Methods can have any name
- Methods can be declared multiple times (overloaded)
 - What makes them different is their signature
- You **cannot** have duplicate signatures (for methods or constructors)!!!
 - Otherwise JVM does not know which one to call when you invoke them

Constructor signatures

```
imagePackage :: RasterImage
```

```
// no visible (public) fields
```

```
// constructors
```

```
+ RasterImage()  
+ RasterImage(int, int)  
+ RasterImage(RasterImage)  
+ RasterImage(String)
```

← constructors
(no return type)

```
// other methods (not all shown)
```

```
+ getGraphics2D() : Graphics2D  
+ getHeight() : int  
+ getWidth() : int  
+ getPixel(int,int) : int  
+ getPixelColor(int, int) : int  
+ setTitle(String) : void  
+ setBasicPixel(int, int, int) : void  
+ show() : void  
+ toString() : String
```

← methods
(have a return type)

Types of methods?

- **Accessors**

- Accesses and returns something relating to the state
 - i.e. a field, computation based on field(s), or representation of field(s)
- Examples:
 - getWidth()** : returns the value of the width of the RasterImage
 - toString()** : returns a string representation of the RasterImage object
 - show()** : displays the RasterImage object as an image in a window
- Special case: “*getter*”
 - a method that returns the value of a specific field defined in the class

- **Mutators**

- Modifies field(s) in the class/object
- Examples:
 - setTitle()** : sets the title of a RasterImage object
 - setBasicPixel(..)** : sets the colour of a pixel using a packed rgb int as an argument
- Special case: “*setter*”
 - a method that sets (modifies) the value of a specific field defined in the class

Method signatures:

`imagePackage :: RasterImage`

`// no visible (public) fields`

`// constructors`

`+ RasterImage()
+ RasterImage(int, int)
+ RasterImage(RasterImage)
+ RasterImage(String)`

← constructors
(no return type)

`// other methods (not all shown)`

`+ getGraphics2D() : Graphics2D
+ getHeight() : int
+ getWidth() : int
+ getPixel(int,int) : int
+ getPixelColor(int, int) : int
+ setTitle(String) : void
+ setBasicPixel(int, int, int) : void
+ show() : void
+ toString() : String`

← methods
(have a return type)

Method signatures:

```
imagePackage :: RasterImage
```

```
// no visible (public) fields
```

```
// constructors
```

```
+ RasterImage()  
+ RasterImage(int, int)  
+ RasterImage(RasterImage)  
+ RasterImage(String)
```

```
// other methods (not all shown)
```

```
+ getGraphics2D() : Graphics2D  
+ getHeight() : int  
+ getWidth() : int  
+ getPixel(int,int) : int  
+ getPixelColor(int, int) : int  
+ setTitle(String) : void  
+ setBasicPixel(int, int, int) : void  
+ show() : void  
+ toString() : String
```

setters & getters imply something about the fields that might be stored in the object

(even if they are private)

Types of methods?

- **Other**

- In RasterImage:

- explore() : causes image to be shown in PictureExplorer window

- repaint() : causes image to redraw itself in the window

- write(..) : causes contents of RasterImage to be written to image file

- **What about these?**

blacken

```
public void blacken(int rowIndex)
```

Mutate the colour of all of the pixels in the specified row to be black. The row index 0 corresponds to the top row. If the passed rowIndex is outside of the row indices that are defined for this image, then nothing happens.

Parameters:

rowIndex - as described above

modifyRowColour

```
public void modifyRowColour(int rowIndex,  
                             java.awt.Color theNewColor)
```

Cause the pixels in the specified row to change to the specified colour. The row index 0 corresponds to the top row. If the passed rowIndex is outside of the row indices that are defined for this image, then nothing happens.

Parameters:

rowIndex - as described above

theNewColor - the new colour of the pixels in the row

ImageExample.java (using RasterImage):

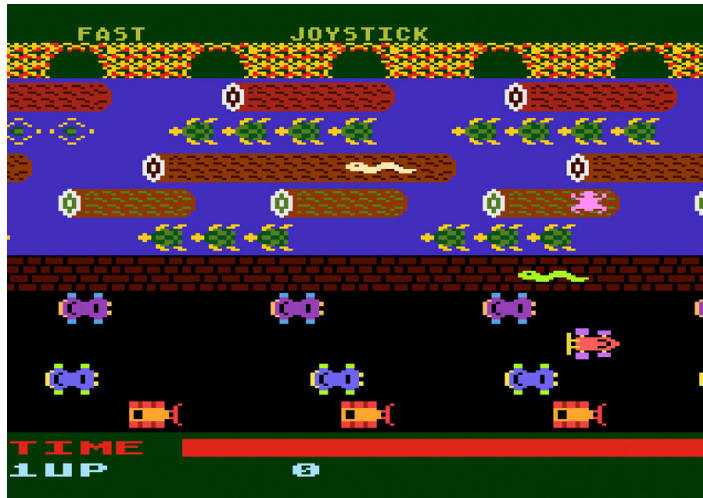
```
import imagePackage.*;  
  
RasterImage img = new RasterImage();  
img = new RasterImage(100,200);  
img = new RasterImage("images/yogaBike.jpg");  
  
img.show();
```



Introduction to Graphics2D

(underlying methods for drawing – yes, these are used/mimicked by processing ultimately)

Frogger !!!



Many versions over the decades !

Lets first assume graphics/image objects are incorporated into a class to create basic components of the backdrop for “frogger” game

- Objects used:
 - **RasterImage**
 - One image that represents the scene
 - **Graphics2D**
 - Within RasterImage, is a Graphics2D object which contains data and methods for drawing
 - `java.awt.Graphics` & `java.awt.Graphics2D` (contain api's for all methods in a Graphics2D object)
 - **Color**
 - Color objects are used to store (define) water and frog colour

Color methods :

`java.awt :: Color`

```
// fields
// lots of static constants (colours) e.g
+ BLACK
+ WHITE
+ RED
```

```
// methods

+ Color(int,int,int)
+ Color(int,int,int,int)
+ Color(float,float,float)
+ Color(float,float,float,float)
+ Color(int)

+ getRed() : int
+ getBlue() : int
+ getGreen() : int
+ getRGB() : int
```

Color class
encapsulates
notion of a colour
(definition & objects)

Has static + dynamic
features

Drawing methods :

`java.awt :: Graphics2D`

`// fields`

`// methods`

`+ setColor(Color)`

`+ drawLine(int, int, int, int)`

`+ drawArc(int, int, int, int, int, int)`

`+ drawRect(int, int, int, int)`

`+ drawOval(int, int, int, int)`

`+ drawPolyline(int[], int[], int)`

`+ drawPolygon(int[], int[], int)`

`+ fillRect(int, int, int, int)`

`+ fillOval(int, int, int, int)`

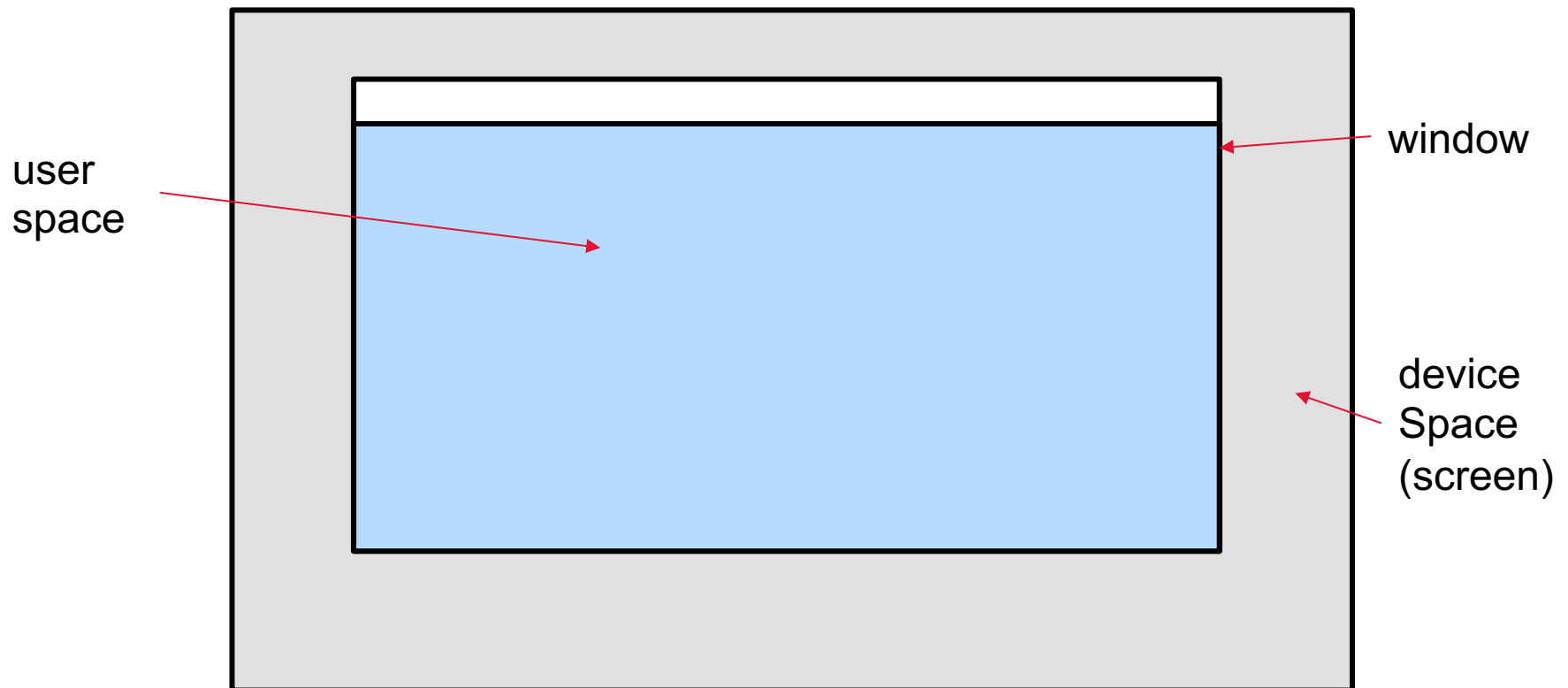
`+`

see `java.awt.Graphics` &
`java.awt.Graphics2D`
for API info

See appendix (of these
slides for more info on
using `Graphics2D`)

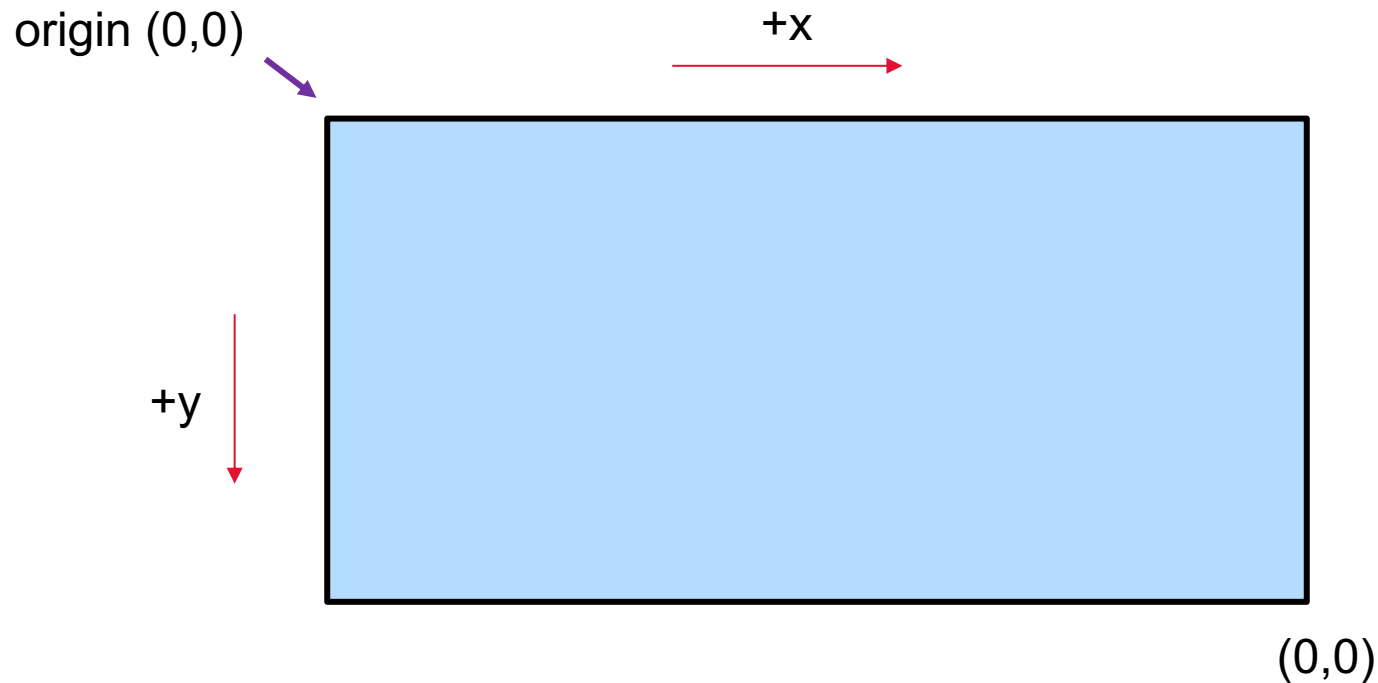
Coordinate System for Java 2D graphics

- User Space => used to specify graphics primitives
- Device Space => coord system of an output device (screen, window, etc.)

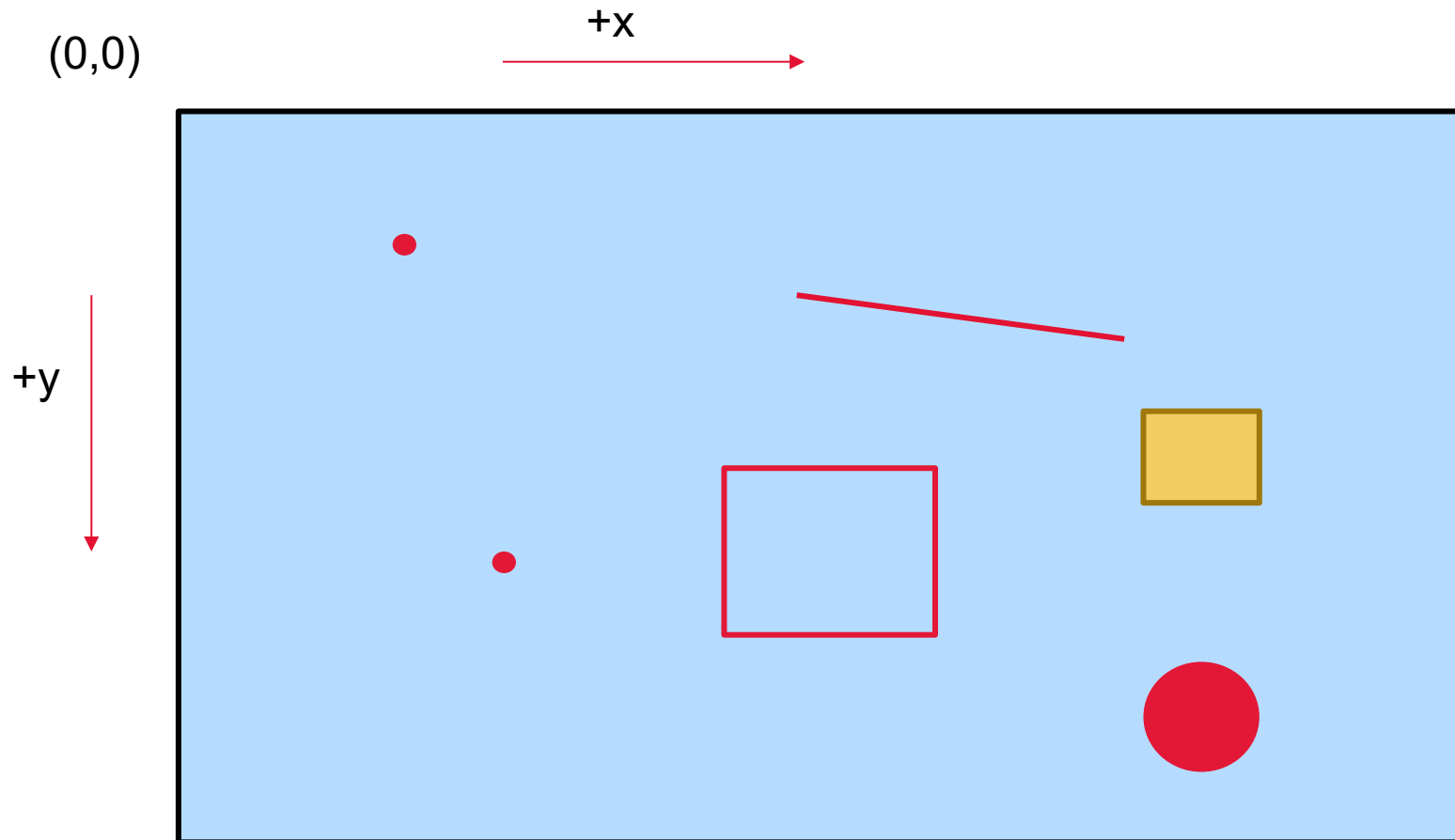


Coordinate System for Java 2D graphics

- User Space (device independent)



Graphics primitives (points/lines/shapes) defined in User Space



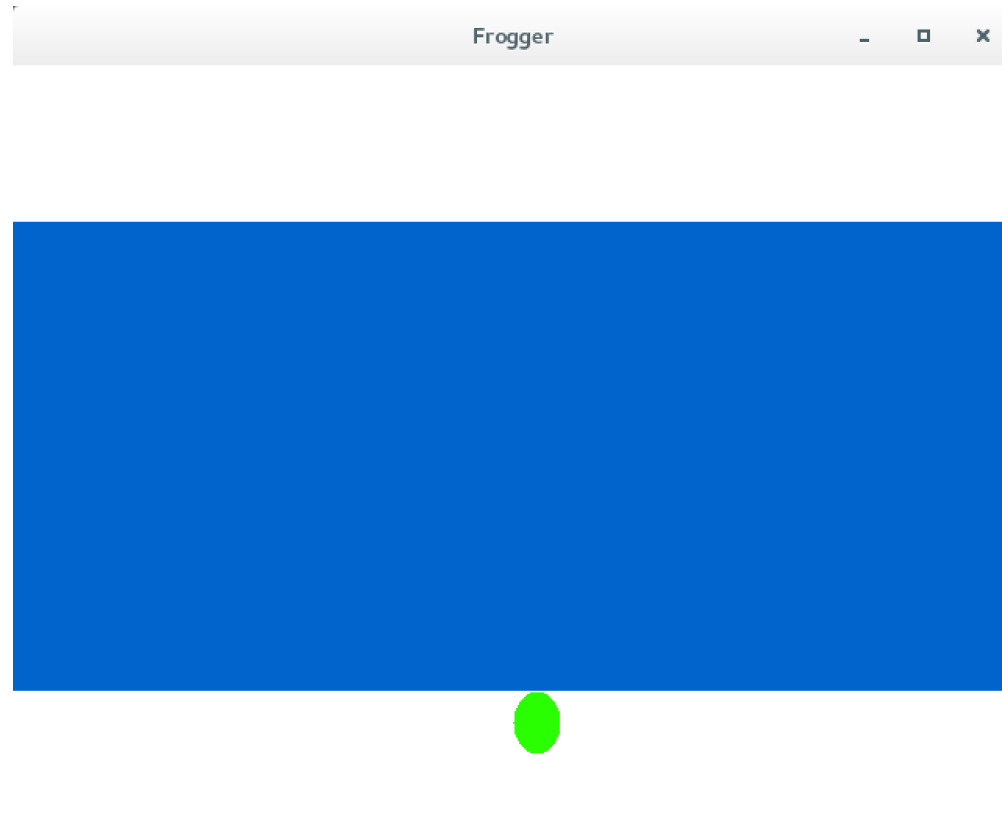
setColor

- Color to be used when drawing objects can be set via **setColor(Color c)**
- Color class has standard Colors (static fields) + constructors (to create specialized colors):

Fields	
Modifier and Type	Field and Description
static Color	black The color black.
static Color	BLACK The color black.
static Color	blue The color blue.
static Color	BLUE The color blue.
static Color	cyan The color cyan.
static Color	CYAN The color cyan.
static Color	DARK_GRAY The color dark gray.
static Color	darkGray The color dark gray.

Constructors
Constructor and Description
Color(ColorSpace cspace, float[] components, float alpha) Creates a color in the specified ColorSpace with the color components specified in the float array and the specified alpha.
Color(float r, float g, float b) Creates an opaque sRGB color with the specified red, green, and blue values in the range (0.0 - 1.0).
Color(float r, float g, float b, float a) Creates an sRGB color with the specified red, green, blue, and alpha values in the range (0.0 - 1.0).
Color(int rgb) Creates an opaque sRGB color with the specified combined RGB value consisting of the red component in bits 16-23, the green component in bits 8-15, and the blue component in bits 0-7.
Color(int rgba, boolean hasalpha) Creates an sRGB color with the specified combined RGBA value consisting of the alpha component in bits 24-31, the red component in bits 16-23, the green component in bits 8-15, and the blue component in bits 0-7.
Color(int r, int g, int b) Creates an opaque sRGB color with the specified red, green, and blue values in the range (0 - 255).
Color(int r, int g, int b, int a) Creates an sRGB color with the specified red, green, blue, and alpha values in the range (0 - 255).

Initial goal: create basic “frogger” scene:



Can draw these shapes directly through Graphics2D

- “Water”

// assume g is a reference to a Graphics2D object

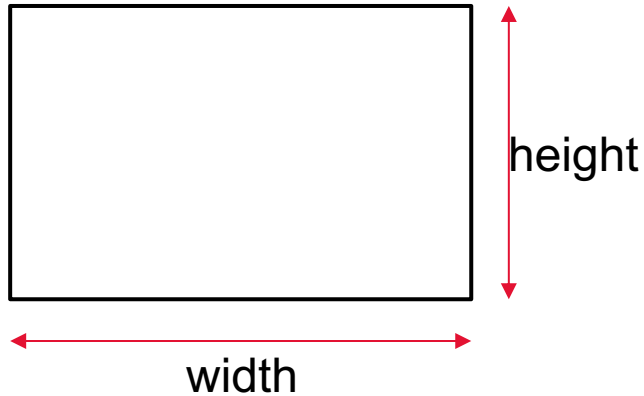
```
g.fillRect(x, y, w, h);           // filled rectangle using  
                                  // current color
```

- “Frog”

// assume g is a reference to a Graphics2D object

```
g.fillOval(x, y, w, h);           // filled ellipse (oval) using  
                                  // current color
```

x,y



drawRect

```
public void drawRect(int x,  
                    int y,  
                    int width,  
                    int height)
```

Draws the outline of the specified rectangle. The left and right edges of the rectangle are at x and $x + \text{width}$. The top and bottom edges are at y and $y + \text{height}$. The rectangle is drawn using the graphics context's current color.

Parameters:

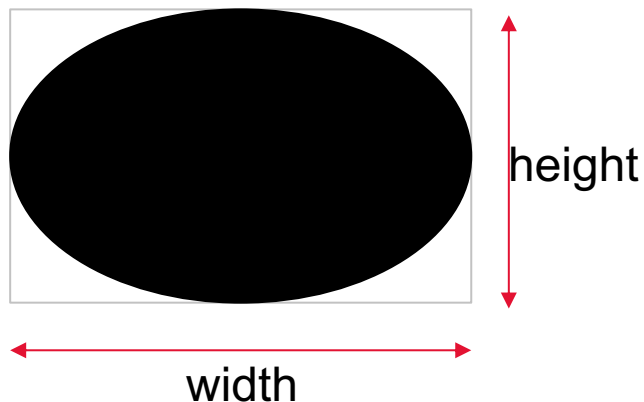
x - the x coordinate of the rectangle to be drawn.

y - the y coordinate of the rectangle to be drawn.

width - the width of the rectangle to be drawn.

height - the height of the rectangle to be drawn.

x,y



fillOval

```
public abstract void fillOval(int x,  
                             int y,  
                             int width,  
                             int height)
```

Fills an oval bounded by the specified rectangle with the current color.

Parameters:

x - the x coordinate of the upper left corner of the oval to be filled.

y - the y coordinate of the upper left corner of the oval to be filled.

width - the width of the oval to be filled.

height - the height of the oval to be filled.

Frogland0: No class fields, main() only:

```
import java.awt.*;           // imports ALL classes from java.awt
import imagePackage.*;       // imports ALL classes from imagePackage

public class Frogland0 {

    public static void main(String[] args) {

        RasterImage frogLand = new RasterImage(640,480);
        Graphics2D gfx = frogLand.getGraphics2D();

        Color frogCol = Color.GREEN;           // use static field
        Color waterCol = new Color(0, 100, 200); // create new color object

        frogLand.show();
        frogLand.setTitle("Frogger");           // RasterImage window

        gfx.setColor(waterCol);
        gfx.fillRect(0, 100, 640, 300);
        gfx.setColor(frogCol);
        gfx.fillOval(320, 400, 30, 40);

    }
}
```

UML for Frogland0 ?

Frogland0
// fields
// methods

A static (utility class) version of this?

- Recall, utility classes hold related data & methods, but cannot be instantiated as objects
 - E.g. Math class:
 - `Math.PI`, `Math.E` [fields]
 - `Math.cos(double angle)`; `Math.abs(int val)`; etc. [methods]

Frogland1 (utility class with main())

```
import java.awt.*;           // imports ALL classes from java.awt
import imagePackage.*;       // imports ALL classes from imagePackage
```

```
public class Frogland1 {

    public static RasterImage frogLand = new RasterImage(640,480);
    public static Graphics2D gfx = frogLand.getGraphics2D();
    public static Color frogCol = Color.GREEN;
    public static Color waterCol = new Color(0, 100, 200);

    public static void main(String[] args) {

        frogLand.show();
        frogLand.setTitle("Frogger");

        gfx.setColor(waterCol);
        gfx.fillRect(0, 100, 640, 300);
        gfx.setColor(frogCol);
        gfx.fillOval(320, 400, 30, 40);

    }
}
```

all features
are static

UML for Frogland1 ?

Frogland1
<pre>// fields + <u>frogLand</u> : RasterImage + <u>gfx</u> : Graphics2D + <u>frogCol</u> : Color + <u>waterCol</u> : Color</pre>
<pre>// methods</pre>

Accessing static fields/methods from a client

```
import java.awt.Color;

public class Client {


    public static void main(String[] args) {

        // we can only access public fields/methods
        // as a client of a particular class!

        Color c = Frogland1.frogCol; // field is public

        System.out.println(c);

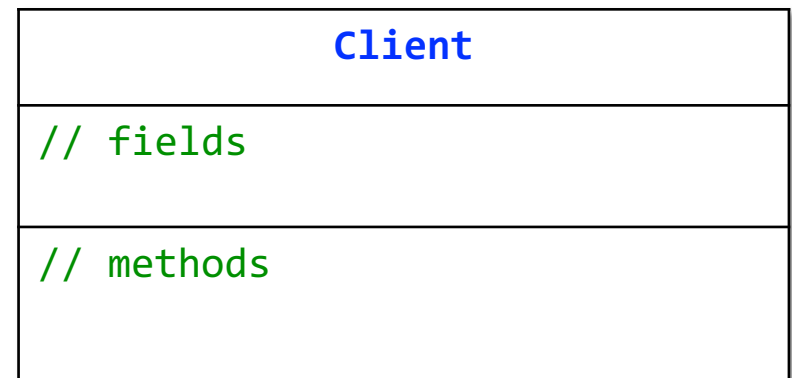
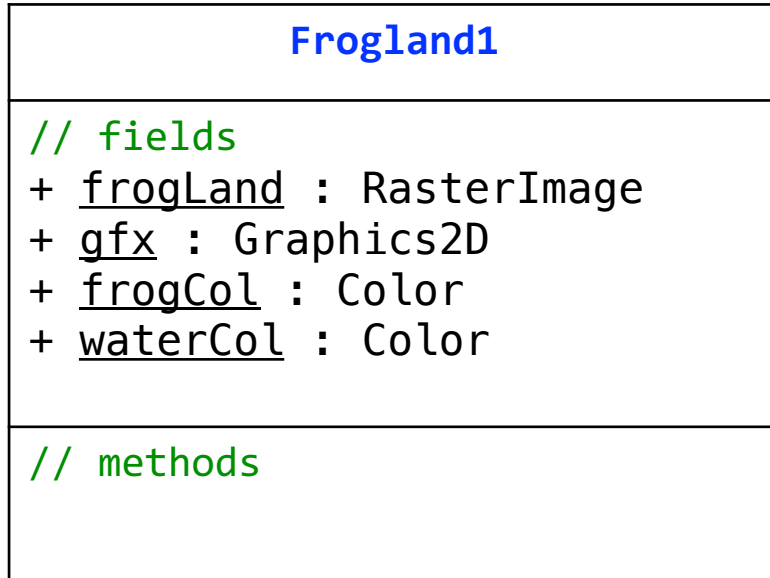
    }
```



Static (utility) class
Static fields/methods must be
accessed through class name

UML?

This class has state
(but no methods)



Accessing class fields from a client

```
import java.awt.Color;

public class Client {

    public static void main(String[] args) {

        // we can only access public fields/methods
        // as a client of a particular class!

        Color c = Frogland1.frogCol; // NOT possible if field is private

        System.out.println(c);

    }
}
```

Class internals

Dynamic Classes (templates for object creation)

Static keyword (the real meaning)

- Fields & Methods may be non-static!
- Static (uses "static" keyword in declaration/header)
 - Means **"only one copy in memory"**
 - Examples:
 - `public static int myInt = 0;`
 - `public static String myStr = "hello";`
 - `public static Point2D p1;`
- Non-static (**no "static" keyword** in declaration/header)
 - Means **"can have multiple copies in memory"** – for objects
 - Usually set when object of this class is *instantiated*
 - Examples:
 - `public int myInt;`
 - `public String myStr;`
 - `public Point2D p1;`

static methods can only
access static fields

non-static methods also
called "instance methods"

Constructors

- the purpose of a constructor:
 - to initialize the state of an object when *instantiated*
 - *it should set the values of all of the non-static fields to appropriate values*
 - *this includes private or public fields..*
- a constructor:
 - must have the same name as the class
 - never returns a value (not even void)
 - constructors are not methods
 - can have zero or more parameters
- 3 fundamental types of constructor
 - **Default; Custom; Copy**

Constructors (3 types)

- **Default Constructor**

- The constructor with no arguments
- If there are fields to initialize, then some predetermined (default settings) are used. API usually indicates defaults

```
+ RasterImage()
```

- **Custom Constructor(s)**

- Any constructor with 1 or more arguments (other than copy constructor)
- Gives multiple ways in which an object (instance) may be initialized by the client

```
+ RasterImage(int, int)  
+ RasterImage(String)
```

- **Copy Constructor**

- A constructor with an argument that is the same type as the object we are creating (i.e. argument is same type as the class)
- Used to create a copy of an existing object of the same type

```
+ RasterImage(RasterImage)
```

Lets organize our class ...
Lets call it ... “Frogland2”

Frogland2

```
// fields  
+ frogLand : RasterImage  
+ gfx : Graphics2D  
+ frogCol : Color  
+ waterCol : Color
```

```
// constructor  
  
+ Frogland2()
```

```
import java.awt.*;           // imports ALL classes from java.awt
import imagePackage.*;       // imports ALL classes from imagePackage
```

```
public class Frogland2 {
```

```
    public static RasterImage frogLand ;
    public static Graphics2D gfx ;
    public static Color frogCol ;
    public static Color waterCol ;
```

```
    public Frogland2() {
        frogLand = new RasterImage(640,480);
        gfx = frogLand.getGraphics2D();
        frogCol = Color.GREEN;
        waterCol = new Color(0, 100, 200);
    }
```

```
    public static void main(String[] args) {
        frogLand.show();
        frogLand.setTitle("Frogger");
        gfx.setColor(waterCol);
        gfx.fillRect(0, 100, 640, 300);
        gfx.setColor(frogCol);
        gfx.fillOval(320, 400, 30, 40);
```

```
    }
}
```

Will this work?

```
import java.awt.*;           // imports ALL classes from java.awt
import imagePackage.*;       // imports ALL classes from imagePackage
```

```
public class Frogland2 {
```

```
    public static RasterImage frogLand ;
    public static Graphics2D gfx ;
    public static Color frogCol ;
    public static Color waterCol ;
```

```
    public Frogland2() {
        frogLand = new RasterImage(640,480);
        gfx = frogLand.getGraphics2D();
        frogCol = Color.GREEN;
        waterCol = new Color(0, 100, 200);
    }
```

```
    public static void main(String[] args) {
        frogLand.show();
        frogLand.setTitle("Frogger");
        gfx.setColor(waterCol);
        gfx.fillRect(0, 100, 640, 300);
        gfx.setColor(frogCol);
        gfx.fillOval(320, 400, 30, 40);
    }
```

```
}
```

Class has not been
instantiated!

Thus ctor not run
fields not initialized

```
import java.awt.*;           // imports ALL classes from java.awt
import imagePackage.*;       // imports ALL classes from imagePackage
```

```
public class Frogland2 {
```

```
    public static RasterImage frogLand ;
    public static Graphics2D gfx ;
    public static Color frogCol ;
    public static Color waterCol ;
```

```
    public Frogland2() {
        frogLand = new RasterImage(640,480);
        gfx = frogLand.getGraphics2D();
        frogCol = Color.GREEN;
        waterCol = new Color(0, 100, 200);
    }
```

```
    public static void main(String[] args) {
```

```
        Frogland2 world = new Frogland2();
```

```
        frogLand.show();
        frogLand.setTitle("Frogger");
        gfx.setColor(waterCol);
        gfx.fillRect(0, 100, 640, 300);
        gfx.setColor(frogCol);
        gfx.fillOval(320, 400, 30, 40);
```

```
    }
```

```
}
```

Instantiated !!

Lets organize our class .. “Frogland2”

Frogland2

```
// fields
+ frogLand : RasterImage
+ gfx : Graphics2D
+ frogCol : Color
+ waterCol : Color
```

```
// constructors
+ Frogland2()
```

```
// methods
+ drawWorld()
```

```
import java.awt.*;
import imagePackage.*;

public class FrogLand2 {

    public RasterImage frogLand;
    public Graphics2D gfx;
    public Color frogCol;
    public Color waterCol;

    public Frogland2() {
        frogLand = new RasterImage(640,480);
        gfx = frogLand.getGraphics2D();
        frogCol = Color.GREEN;
        waterCol = new Color(0, 100, 200);
    }

    public void drawWorld() {
        frogLand.show();
        frogLand.setTitle("Frogger");
        gfx.setColor(waterCol);
        gfx.fillRect(0, 100, 640, 300);
        gfx.setColor(frogCol);
        gfx.fillOval(320, 400, 30, 40);
    }

    // main shown on next page ...
}
```

```
// ...
```

```
public static void main(String[] args) {
```

```
    Frogland2 world = new Frogland2();
```

```
    // public method drawWorld allows a client to create the graphics  
    // without having to do it directly themselves.
```

```
    world.drawWorld();
```

```
}
```

```
}
```


Topics:

- Encapsulating Frogland?
 - Make fields non-static?
 - Adding custom constructors
 - Making fields private
 - Add public getters/setters where access is needed
 - Adding other methods
- More on Graphics2D (`java.awt.geom`)

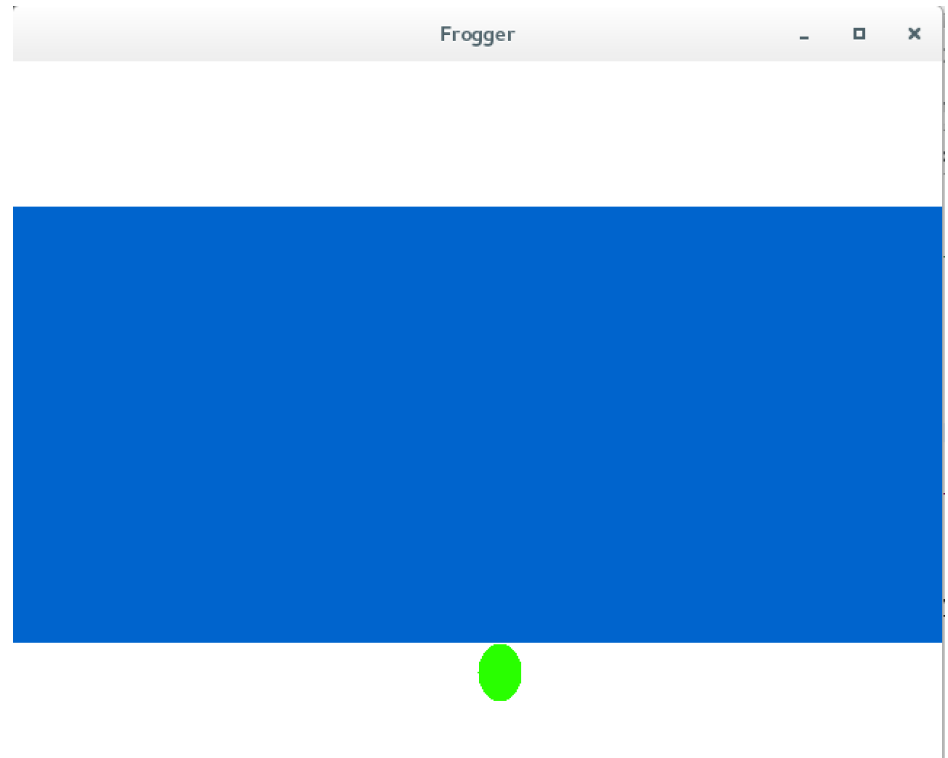
Recall: using Graphics2D methods to create a basic “frogger” scene

Frogland2

```
// fields
+ frogLand : RasterImage
+ gfx : Graphics2D
+ frogCol : Color
+ waterCol : Color
```

```
// constructors
+ Frogland2()
```

```
// methods
+ drawWorld()
```



So far, we have created a class that has some static properties (fields), and some non-static properties (default constructor + draw method)

What if the fields were made non-static?

- If not declared static, these become instance fields
- Static fields
 - “one copy in memory”
 - Not one copy per object; rather: one copy per class
 - Accessed via class name (not object name)
- Instance fields
 - “one unique copy of field per object (instance)”

static vs. non-static

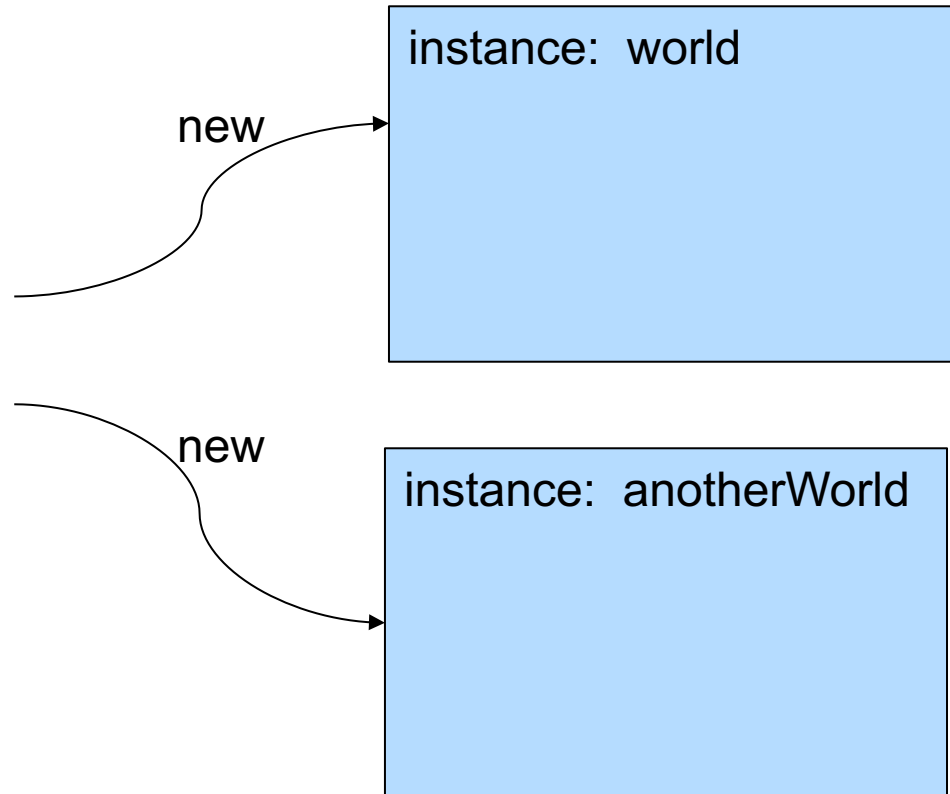
```
// consider two instances created:
```

```
public static void main(String[] args) {  
    Frogland2 world = new Frogland2();  
    Frogland2 anotherWorld = new Frogland2();  
    // ...  
}
```

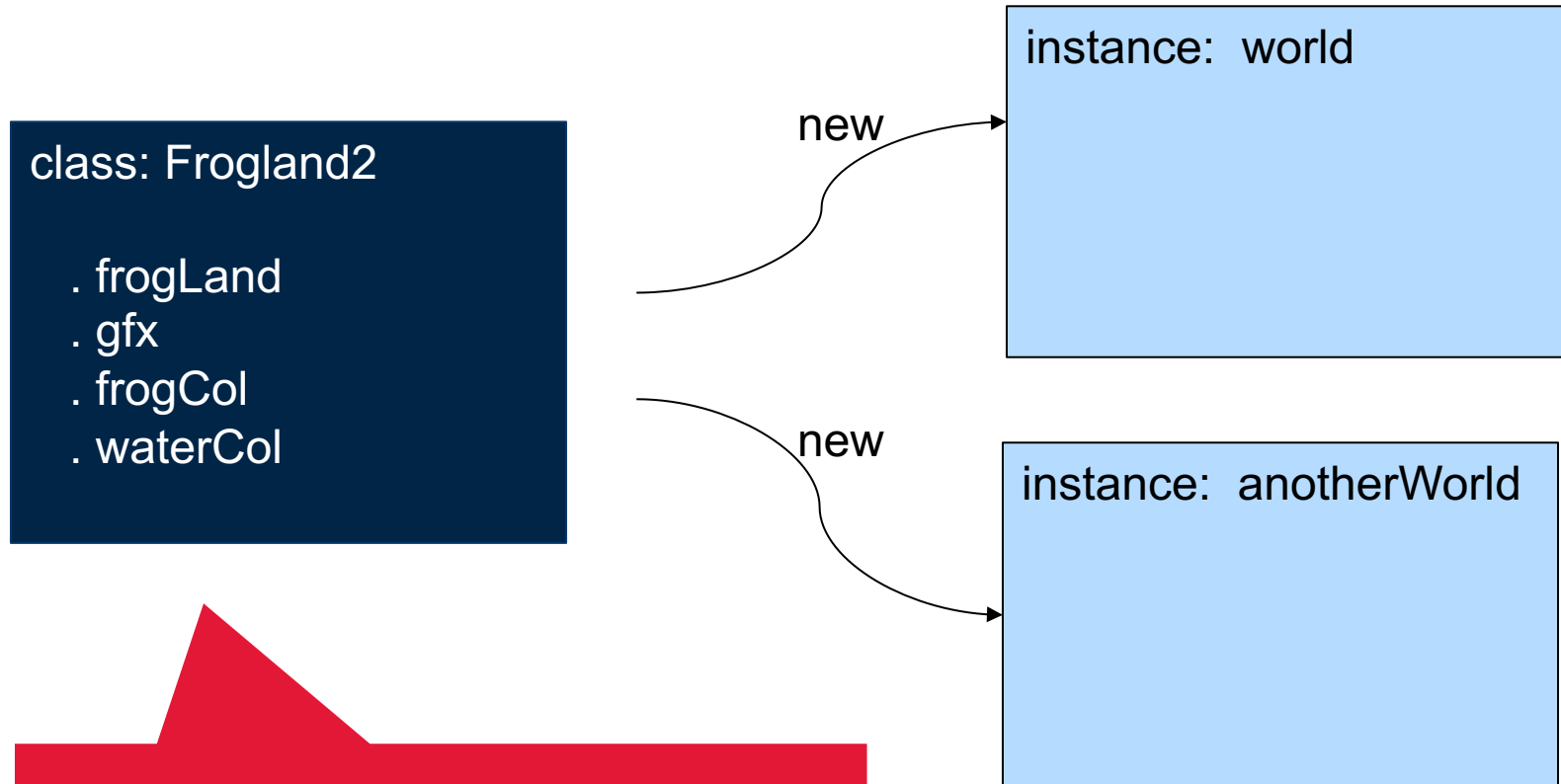
static fields

```
class: Frogland2  
  
. frogLand  
. gfx  
. frogCol  
. waterCol
```

static fields
stored in class
area of memory

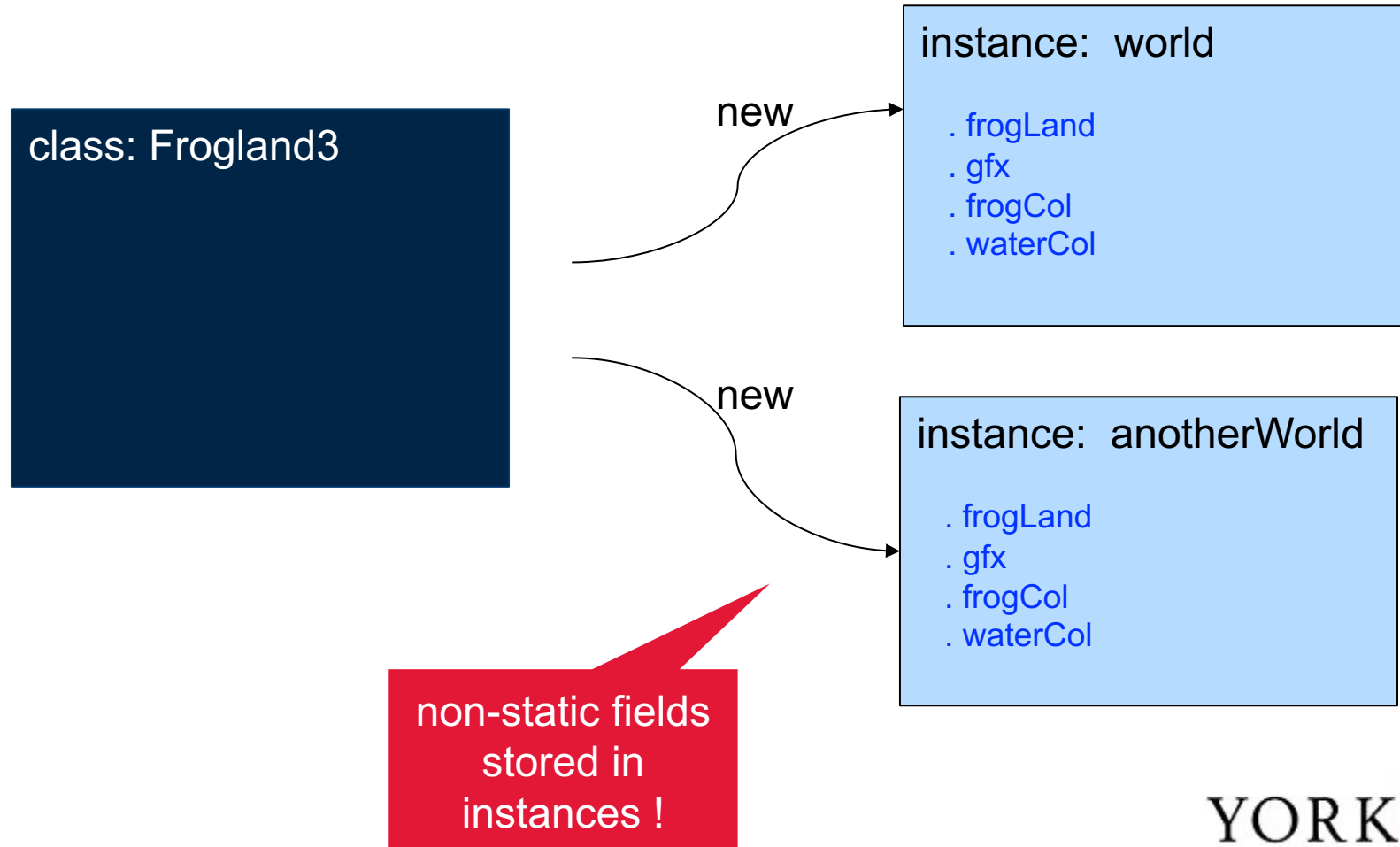


static fields

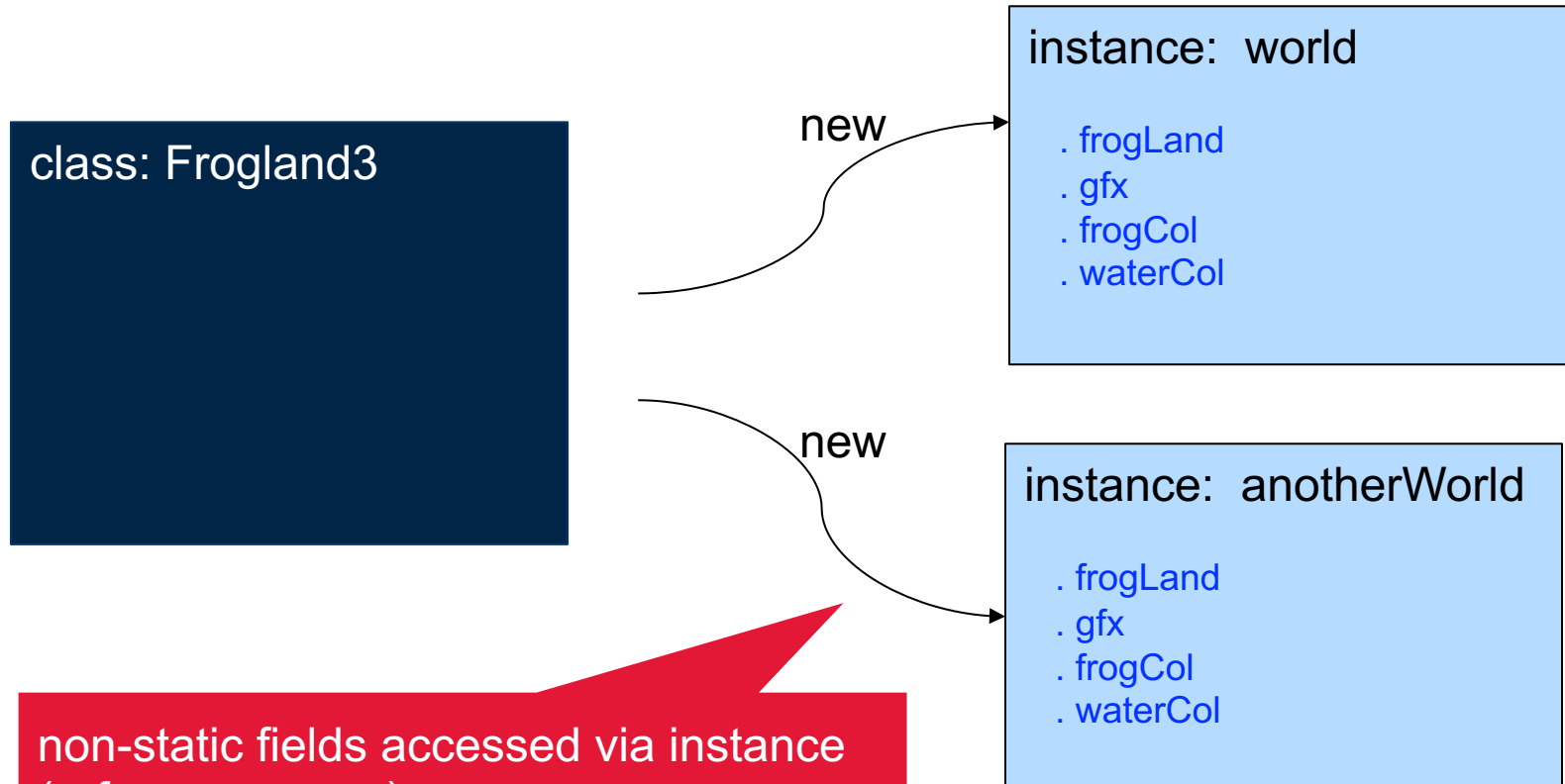


static fields accessed via class name:
e.g. `Frogland2.frogLand`

non-static fields



non-static fields



non-static fields accessed via instance
(reference name):

e.g. `world.frogLand`
`anotherWorld.frogLand`


```
import java.awt.*;  
import imagePackage.*;
```

```
public class FrogLand3 {
```

```
    public RasterImage frogLand;  
    public Graphics2D gfx;  
    public Color frogCol;  
    public Color waterCol;
```

non-static

```
    public FrogLand3() {  
        frogLand = new RasterImage(640,480);  
        gfx = frogLand.getGraphics2D();  
        frogCol = Color.GREEN;  
        waterCol = new Color(0, 100, 200);  
    }
```

constructor
initializes
each
instance

```
    public void drawWorld() {  
        frogLand.show();  
        frogLand.setTitle("Frogger");  
        gfx.setColor(waterCol);  
        gfx.fillRect(0, 100, 640, 300);  
        gfx.setColor(frogCol);  
        gfx.fillOval(320, 400, 30, 40);  
    }
```

```
// main shown on next page ...
```

```
// ...
```

```
public static void main(String[] args) {
```

```
FrogLand3 world = new FrogLand3();
```

```
FrogLand3 anotherWorld = new FrogLand3();
```

```
// public method drawWorld allows a client to create the graphics  
// without having to do it directly themselves.
```

```
world.drawWorld();
```

```
}
```

```
}
```

Constructors revisited..

- Constructor(s):
 - Allows many instances of FrogLand3 to be created
 - But currently we can only initialize one way !
 - Any constructor should ensure ALL fields are initialized!
 - With static fields, all of these instances share these same fields (in memory)
 - With non-static fields, each instance has its own unique field values
- We need to add some custom constructors to set unique values for these fields when these instances are created!

Custom constructors

- A class can have multiple constructors, but the signatures of the constructors must be unique
- I.e. each constructor must have a unique list of parameter types
- E.g. it would be convenient for clients if FrogLandX could have a constructor to set the following properties:
 - Size of RasterImage
 - Initial frog colour
 - Initial water colour

Custom constructors

Frogland4

```
// fields
```

```
+ frogLand : RasterImage  
+ gfx : Graphics2D  
+ frogCol : Color  
+ waterCol : Color
```

```
// constructors
```

```
+ Frogland4()  
+ Frogland4(int, int)  
+ Frogland4(int, int, Color)  
+ Frogland4(int, int, Color, Color)
```

```
// methods
```

```
+ drawWorld()
```

```
import java.awt.*;
import imagePackage.*;
```

```
public class Frogland4 {
```

```
    public RasterImage frogLand;
    public Graphics2D gfx;
    public Color frogCol;
    public Color waterCol;
```

```
    public Frogland4() {
        frogLand = new RasterImage(640,480);
        gfx = frogLand.getGraphics2D();
        frogCol = Color.GREEN;
        waterCol = new Color(0, 100, 200);
    }
```

Custom
constructors

```
    public Frogland4(int width, int height) {
        frogLand = new RasterImage(width,height);
        gfx = frogLand.getGraphics2D();
        frogCol = Color.GREEN;
        waterCol = new Color(0, 100, 200);
    }
```

Sets RasterImage to
custom size
(other fields are defaults)

```
    public Frogland4(int width, int height, Color fcol) {
        frogLand = new RasterImage(width,height);
        gfx = frogLand.getGraphics2D();
        frogCol = fcol;
        waterCol = new Color(0, 100, 200);
    }
```

Sets RasterImage to
custom size & frogCol
(other fields are defaults)

```
    public Frogland4(int width, int height, Color fcol, Color wcol) {
        frogLand = new RasterImage(width,height);
        gfx = frogLand.getGraphics2D();
        frogCol = fcol;
        waterCol = wcol
    }
```

Sets RasterImage to custom
size, frogCol & waterCol

the “this” reference

- Within a class constructor or instance method, the keyword “this” is an implicit reference to the current object
 - i.e. keyword “this” is a reference to the:
 - address of object being created if constructor is run, or
 - address of object method is invoked on
 - Generally used to disambiguate between constructor/method parameter name and the class field (if same name used)
 - Good practice to use **this.fieldname** for any use of/assignment to a class field, and **fieldname** for the parameter

```
import java.awt.*;
import imagePackage.*;
```

```
public class FrogLand4 {
```

```
    public RasterImage frogLand;
    public Graphics2D gfx;
    public Color frogCol;
    public Color waterCol;
```

using "this" reference

```
    public FrogLand4() {
        this.frogLand = new RasterImage(640,480);
        this.gfx = this.frogLand.getGraphics2D();
        this.frogCol = Color.GREEN;
        this.waterCol = new Color(0, 100, 200);
    }
```

```
    public FrogLand4(int width, int height) {
        this.frogLand = new RasterImage(width,height);
        this.gfx = this.frogLand.getGraphics2D();
        this.frogCol = Color.GREEN;
        this.waterCol = new Color(0, 100, 200);
    }
```

```
    public FrogLand4(int width, int height, Color frogCol) {
        this.frogLand = new RasterImage(width,height);
        this.gfx = this.frogLand.getGraphics2D();
        this.frogCol = frogCol;
        this.waterCol = new Color(0, 100, 200);
    }
```

this.frogCol (class field)
frogCol (method parameter)

```
    public FrogLand4(int width, int height, Color frogCol,
        Color waterCol) {
        this.frogLand = new RasterImage(width,height);
        this.gfx = this.frogLand.getGraphics2D();
        this.frogCol = frogCol;
        this.waterCol = waterCol;
    }
```

frogCol and waterCol are
method parameters..

Adding accessors/mutators.. “FroglandLite4”

Frogland5

// fields

- frogLand : RasterImage
- gfx : Graphics2D
- frogCol : Color
- waterCol : Color

// constructors (not shown)

// methods

- + getFrogCol() : Color
- + getWaterCol() : Color
- + setFrogCol(Color)
- + setWaterCol(Color)
-
- + drawWorld()

restrict access

Provide access to
private fields (through
getters/setters)
== ENCAPSULATION

Any access allowed
becomes public API

```
import java.awt.*;
import imagePackage.*;

public class FrogLand5 {

    private RasterImage frogLand;
    private Graphics2D gfx;
    private Color frogCol;
    private Color waterCol;

    public FrogLand5() {
        //not shown
    }
    public FrogLand5(int width, int height) {
        //not shown
    }
    public FrogLand5(int width, int height, Color frogCol) {
        //not shown
    }
    public FrogLand5(int width, int height, Color frogCol, Color waterCol) {
        //not shown
    }

    public void drawWorld() {
        // not shown
    }

    // more on next page ...
}
```

```
// ...
```

```
public Color getFrogCol() {  
    return this.frogCol ;  
}
```

```
public Color getWaterCol() {  
    return this.waterCol ;  
}
```

```
public void setFrogCol(Color frogCol) {  
    this.frogCol = frogCol;  
}
```

```
public void Color setWaterCol(Color waterCol) {  
    this.waterCol = waterCol ;  
}
```

```
public static void main(String[] args) {
```

```
    FrogLand5 world = new FrogLand5();  
    world.drawWorld();
```

```
    FrogLand5 world2 = new FrogLand5();  
    world.drawWorld();
```

```
}
```

```
}
```

Not static methods
Why?

Only instance
methods can access
instance fields..

Notes:

- What are the issues with the following?

```
public FrogLand4(int width, int height, Color frogCol, Color waterCol) {  
    this.frogLand = new RasterImage(this.width,this.height);  
    this.gfx = this.frogLand.getGraphics2D();  
    frogCol = frogCol;  
    waterCol = waterCol;  
}  
  
public void setFrogCol(Color frogCol) {  
    frogCol = frogCol;  
}  
  
public void Color getWaterCol(Color waterCol) {  
    waterCol = waterCol ;  
}
```

Adding more functionality?

FroglandX

```
// fields
- frogLand : RasterImage
- gfx : Graphics2D
- frogCol : Color
- waterCol : Color
```

// constructors

```
+ FroglandX()
```

// methods

```
+ getFrogCol() : Color
+ getWaterCol() : Color
+ setFrogCol(Color)
+ setWaterCol(Color)
+ drawWorld()
```

```
+ moveFrog(...)?
```

Right now, position of water and frog not saved in the state

java.awt.geom package defines some geometric shapes that could be used: e.g. Rectangle2D

How about some methods to modify the position of the frog?

Redrawing also becomes a factor after each modification ...
use *this.gfx.repaint()*

Appendix

more on Graphics2D & java.awt.geom

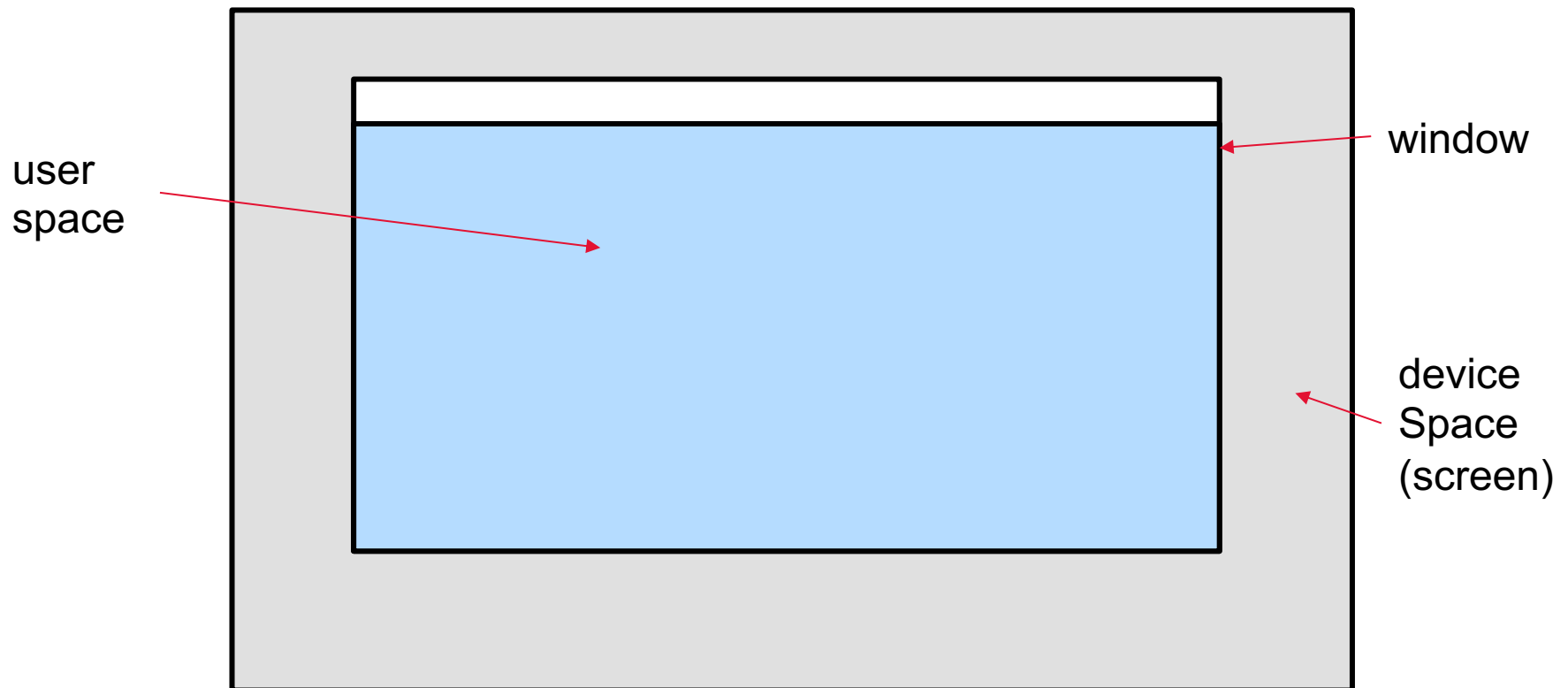
More on Graphics2D and java.awt.geom

- Drawing points/lines/arcs
- Shapes
 - Rectangle, Ellipse
 - YU logo
 - Scaling
 - Polygons
- Drawing Shapes

- java.awt
 - abstract window toolkit (awt)
 - provides classes necessary to create user interfaces and for painting graphics/images
- java.awt.geom
 - provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry
- JAVA 2D API Tutorial/Overview
 - <https://docs.oracle.com/javase/tutorial/2d/overview/index.html>

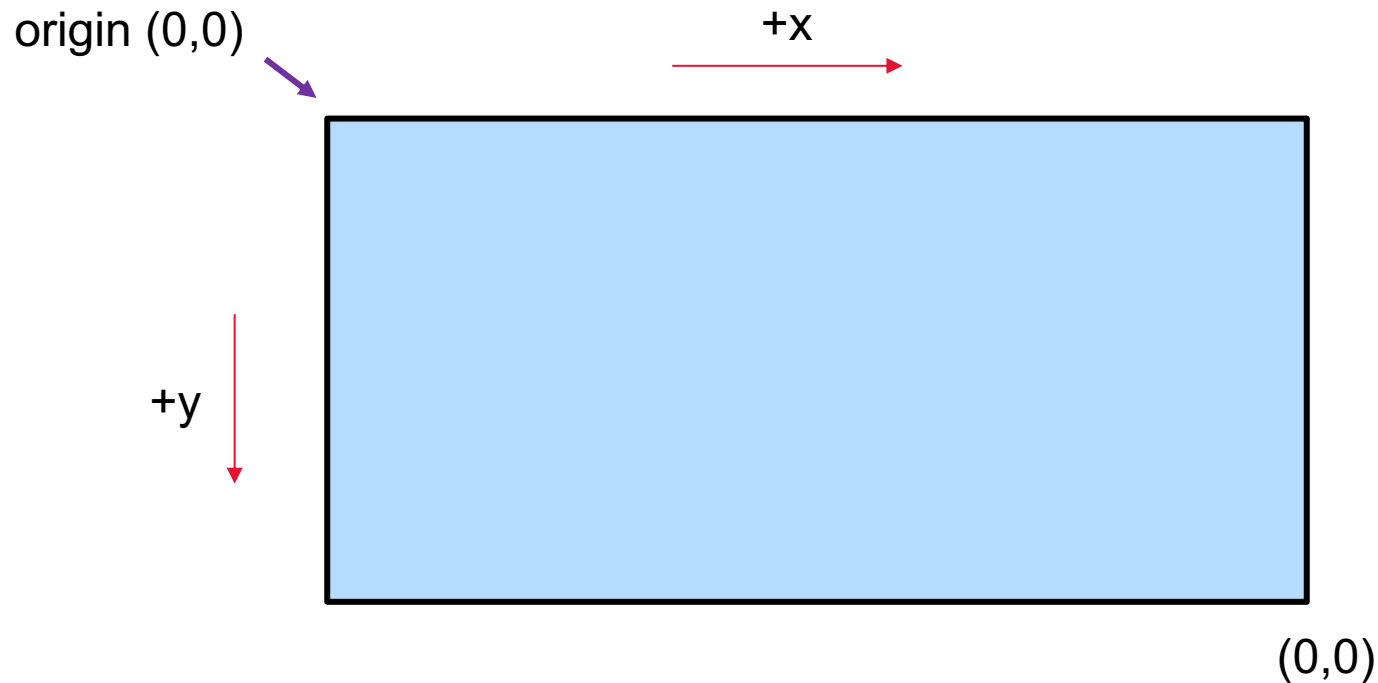
Coordinate System for Java 2D graphics

- User Space => used to specify graphics primitives
- Device Space => coord system of an output device (screen, window, etc.)

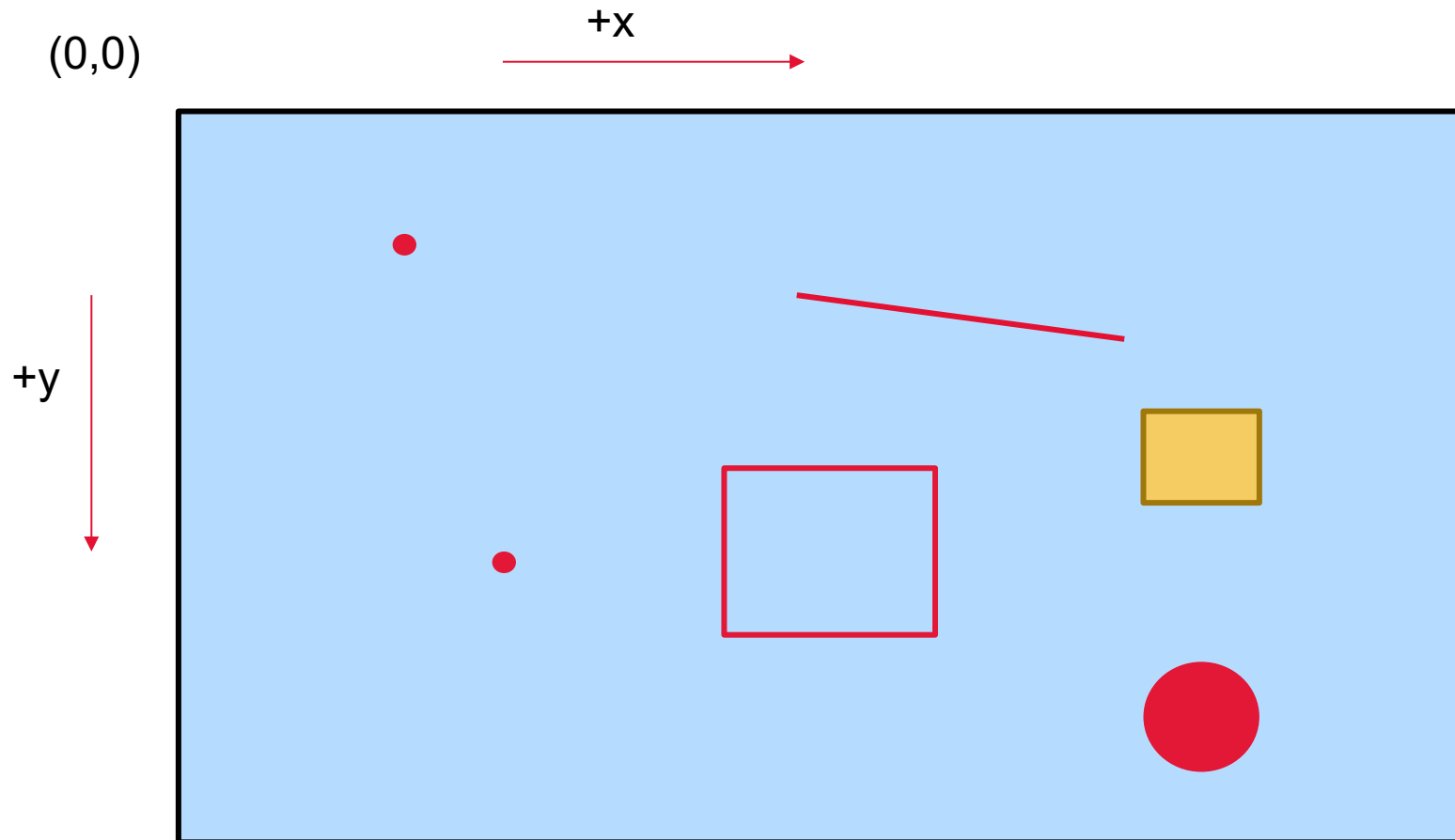


Coordinate System for Java 2D graphics

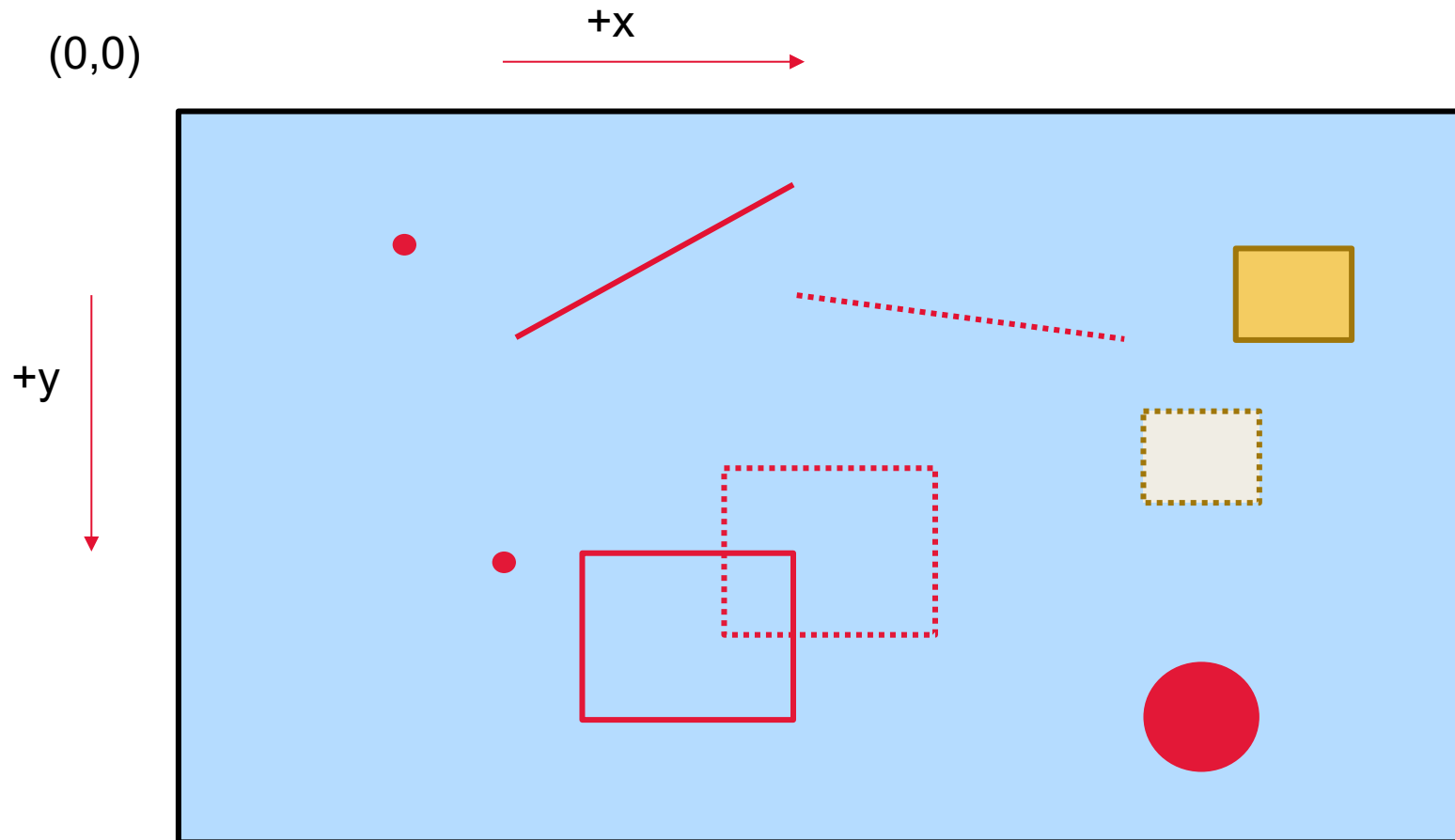
- User Space (device independent)



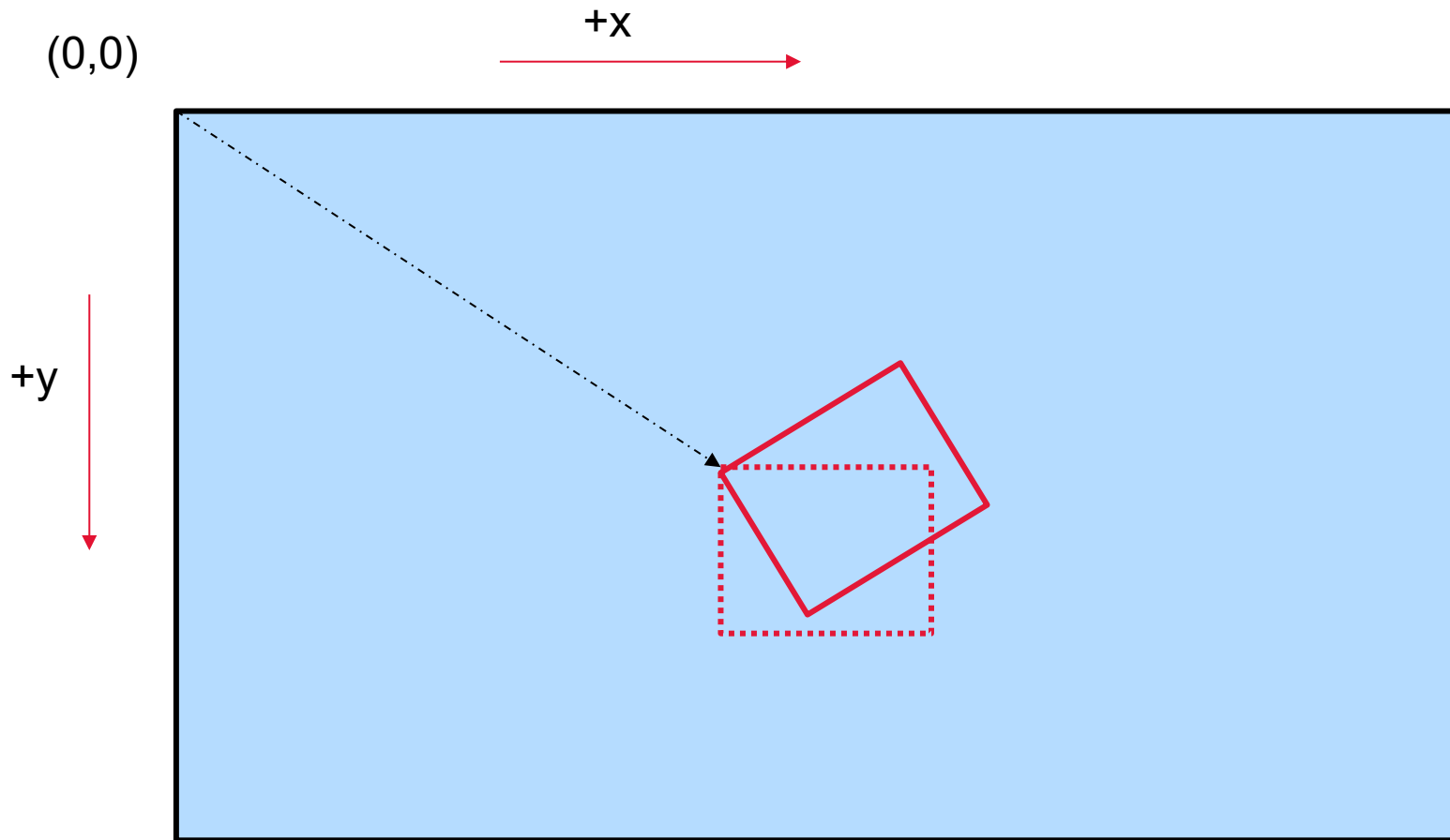
Graphics primitives (points/lines/shapes) defined in User Space



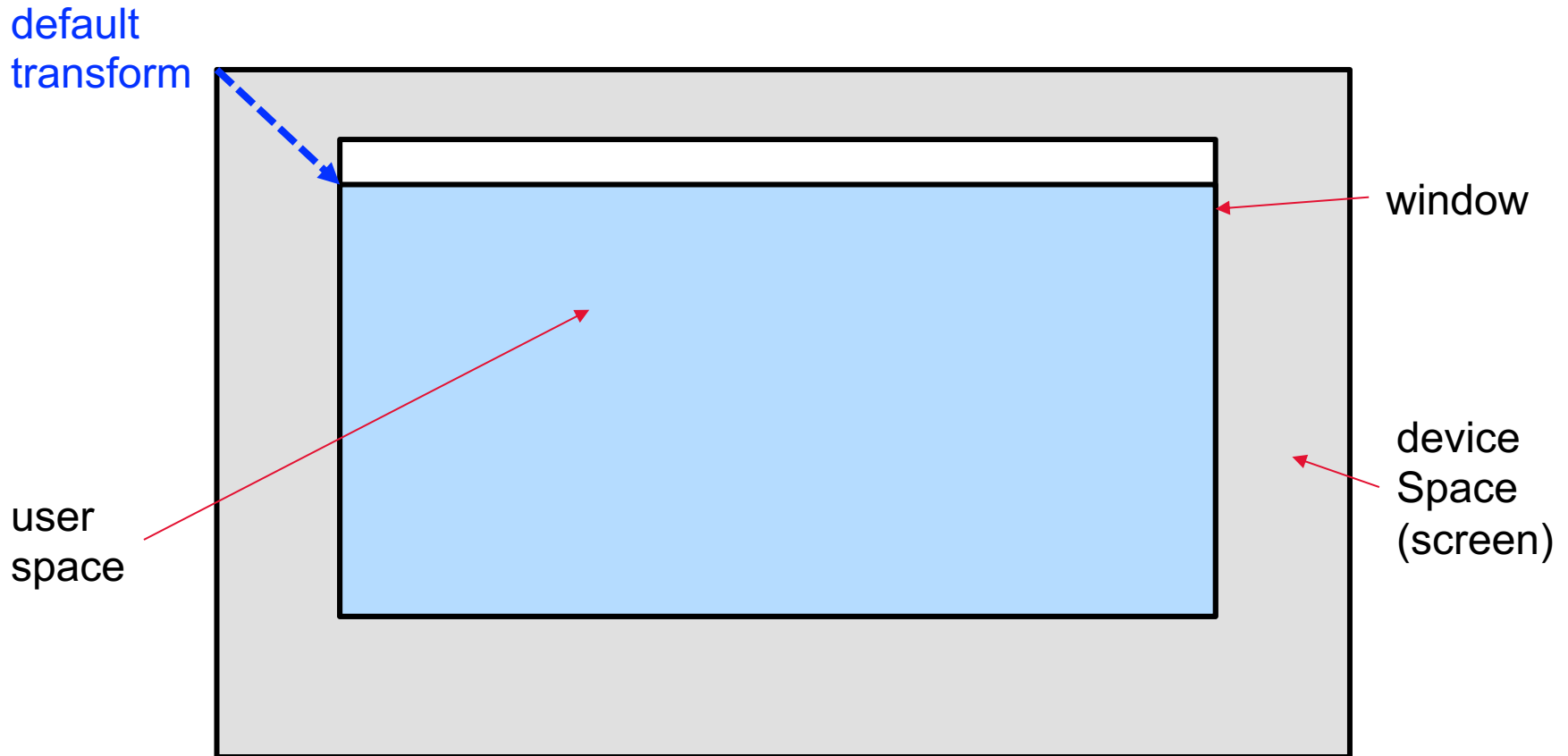
primitives can be “moved” by clearing screen, repositioning and redrawing.. OR.. by using transforms (more convenient)



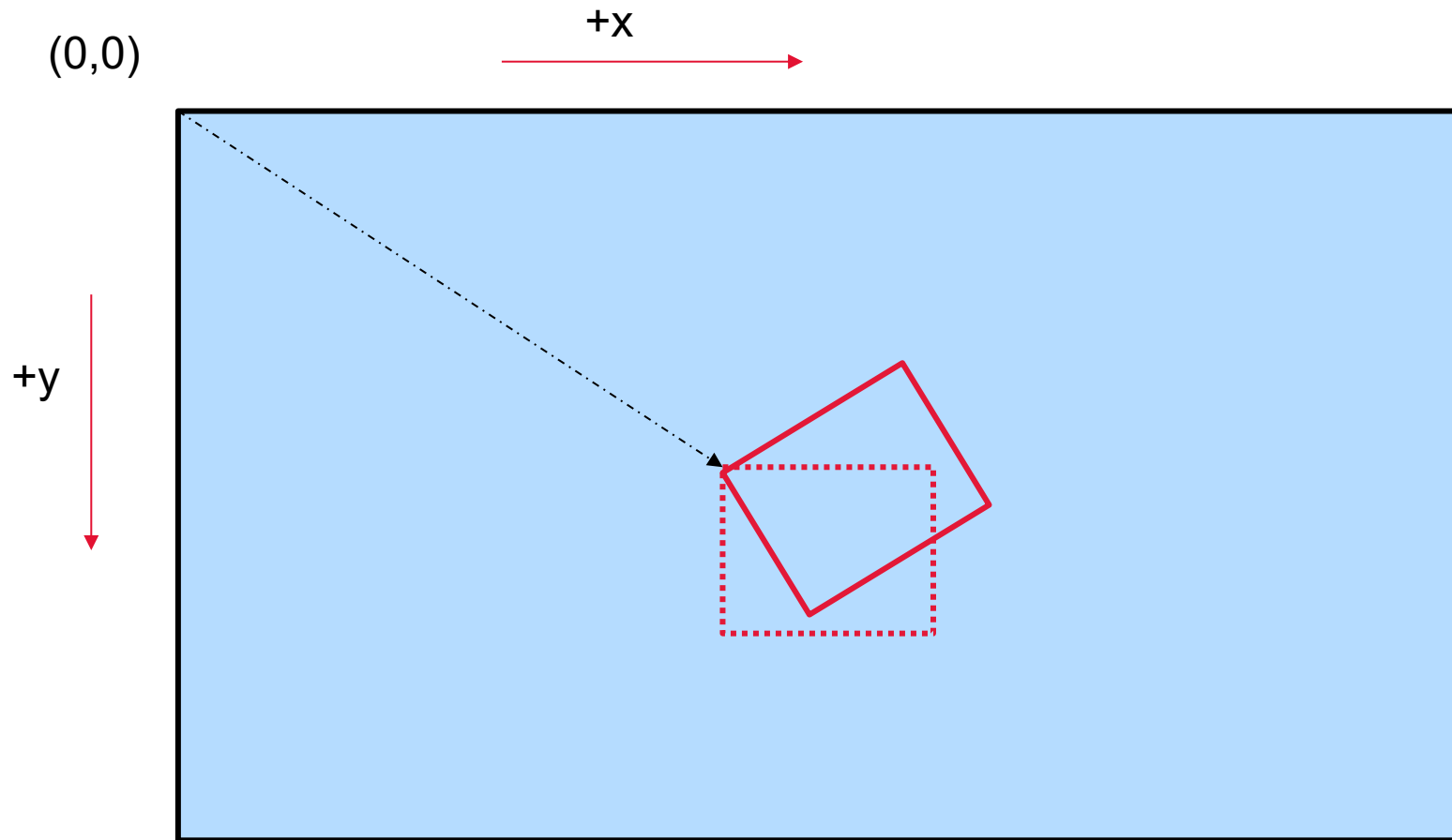
All graphics primitives have a Transform field:
positioning relative to origin: (translation + rotation)



- User Space also positioned inside Device Space via a Transform (the default transform)



Positioned relative to Origin via a Transform: (translation + rotation)



Example

// assume g is a Graphics2D reference

- Clearing screen:
 - g.clearRect(int x, int y, int width, int height)

// drawing is achieved via many draw methods

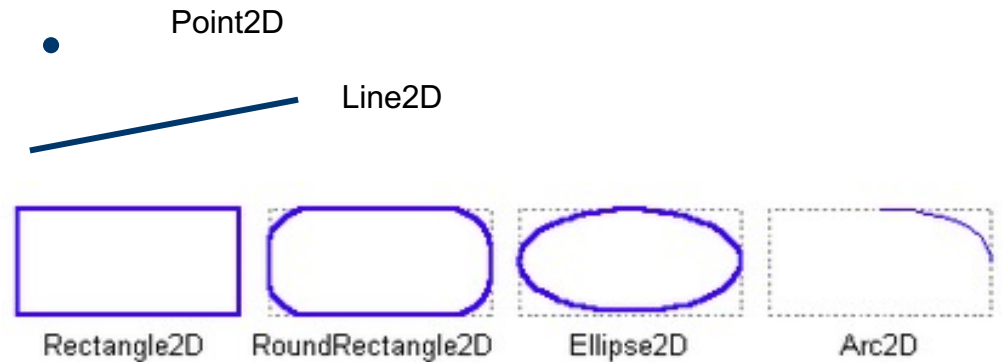
- Some defined directly in Graphics/Graphics2D

Rendering

- Rendering of geometric primitives on screen is achieved using Stroke and Paint attributes
 - Stroke defines thickness and style of how points/lines are drawn (think of it as pen)
 - Paint defines how color patterns can be generated (Paint is a bit more involved than Color, as it allows textures to be drawn not just colors)
- Before a draw operation (there are many draw operations..)
 - **Use Graphics2D's mutators to setup stroke/paint**
 - **setColor(Color c);**
 - **setStroke(Stroke s);**

Geometric Primitives

- Points
- Lines
- Rectangular Shapes



- More complex curves



- Arbitrary shapes (Paths)



Note

- Many of the classes in **java.awt.geom** use nested classes to allow for positions/sizes to be expressed using different primitive fields

The Point2D class defines a point representing a location in (x,y) coordinate space.

This class is only the abstract superclass for all objects that store a 2D coordinate. The actual storage coordinates is left to the subclass.

Since:

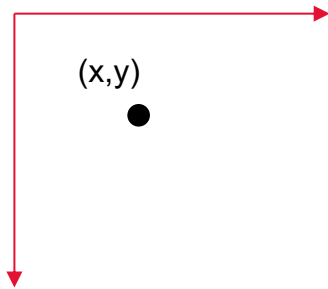
1.2

Nested Class Summary	
Nested Classes	
Modifier and Type	Class and Description
static class	Point2D.Double The Double class defines a point specified in double precision.
static class	Point2D.Float The Float class defines a point specified in float precision.

- E.g. Point2D (has 2 internal classes: Double & Float)
 - To use one vs. the other, you use the inner class via the reference to the outer class

Point2D.Double

(class in java.awt.geom)



Field Summary

Fields

Modifier and Type	Field and Description
double	x The X coordinate of this Point2D.
double	y The Y coordinate of this Point2D.

Constructor Summary

Constructors

Constructor and Description

Double()

Constructs and initializes a Point2D with coordinates (0, 0).

Double(double x, double y)

Constructs and initializes a Point2D with the specified coordinates.

Method Summary

All Methods

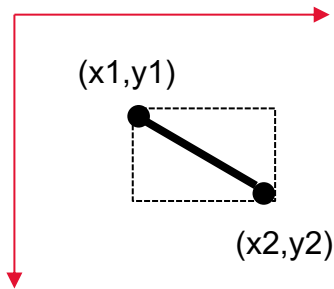
Instance Methods

Concrete Methods

Modifier and Type	Method and Description
double	getX() Returns the X coordinate of this Point2D in double precision.
double	getY() Returns the Y coordinate of this Point2D in double precision.
void	setLocation(double x, double y) Sets the location of this Point2D to the specified double coordinates.
String	toString() Returns a String that represents the value of this Point2D.

Line2D.Double

(class in java.awt.geom)



also .. getX2() getY1(), getY2() =>

Fields

Modifier and Type

Field and Description

double

x1

The X coordinate of the start point of the line segment.

double

x2

The X coordinate of the end point of the line segment.

double

y1

The Y coordinate of the start point of the line segment.

double

y2

The Y coordinate of the end point of the line segment.

Constructors

Constructor and Description

Double()

Constructs and initializes a Line with coordinates (0, 0) → (0, 0).

Double(double x1, double y1, double x2, double y2)

Constructs and initializes a Line2D from the specified coordinates.

Double(Point2D p1, Point2D p2)

Constructs and initializes a Line2D from the specified Point2D objects.

All Methods

Instance Methods

Concrete Methods

Modifier and Type

Method and Description

Rectangle2D

getBounds2D()

Returns a high precision and more accurate bounding box of the Shape than the getBounds method.

Point2D

getP1()

Returns the start Point2D of this Line2D.

Point2D

getP2()

Returns the end Point2D of this Line2D.

double

getX1()

Returns the X coordinate of the start point in double precision.

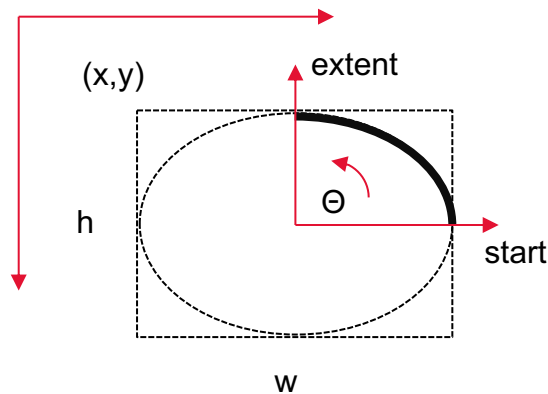
void

setLine(double x1, double y1, double x2, double y2)

Sets the location of the end points of this Line2D to the specified double coordinates.

Arc2D.Double

(class in java.awt.geom)



Fields

Modifier and Type	Field and Description
double	extent The angular extent of the arc in degrees.
double	height The overall height of the full ellipse of which this arc is a partial section (not considering the angular extents).
double	start The starting angle of the arc in degrees.
double	width The overall width of the full ellipse of which this arc is a partial section (not considering the angular extents).
double	x The X coordinate of the upper-left corner of the framing rectangle of the arc.
double	y The Y coordinate of the upper-left corner of the framing rectangle of the arc.

Constructor Summary

Constructors

Constructor and Description

Double()

Constructs a new OPEN arc, initialized to location (0, 0), size (0, 0), angular extents (start = 0, extent = 0).

Double(double x, double y, double w, double h, double start, double extent, int type)

Constructs a new arc, initialized to the specified location, size, angular extents, and closure type.

Double(int type)

Constructs a new arc, initialized to location (0, 0), size (0, 0), angular extents (start = 0, extent = 0), and the specified closure type.

Double(Rectangle2D ellipseBounds, double start, double extent, int type)

Constructs a new arc, initialized to the specified location, size, angular extents, and closure type.

All Methods

Instance Methods

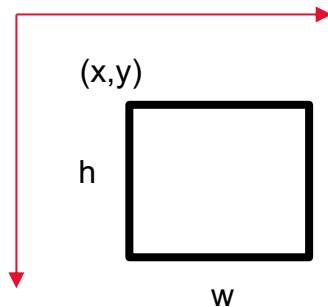
Concrete Methods

Modifier and Type	Method and Description
double	getAngleExtent() Returns the angular extent of the arc.
double	getAngleStart() Returns the starting angle of the arc.

void	setAngleExtent(double angExt) Sets the angular extent of this arc to the specified double value.
void	setAngleStart(double angSt) Sets the starting angle of this arc to the specified double value.
void	setArc(double x, double y, double w, double h, double angSt, double angExt, int closure) Sets the location, size, angular extents, and closure type of this arc to the specified double values.

Rectangle2D.Double

(class in java.awt.geom)



Fields

Modifier and Type	Field and Description
double	height The height of this Rectangle2D.
double	width The width of this Rectangle2D.
double	x The X coordinate of this Rectangle2D.
double	y The Y coordinate of this Rectangle2D.

Constructors

Constructor and Description
Double() Constructs a new Rectangle2D, initialized to location (0, 0) and size (0, 0).
Double(double x, double y, double w, double h) Constructs and initializes a Rectangle2D from the specified double coordinates.

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method and Description
Rectangle2D	getBounds2D() Returns a high precision and more accurate bounding box of the Shape than the getBounds method.
double	getHeight() Returns the height of the framing rectangle in double precision.
double	getWidth() Returns the width of the framing rectangle in double precision.
double	getX() Returns the X coordinate of the upper-left corner of the framing rectangle in double precision.
double	getY() Returns the Y coordinate of the upper-left corner of the framing rectangle in double precision.
boolean	isEmpty() Determines whether the RectangularShape is empty.

void	setRect(double x, double y, double w, double h) Sets the location and size of this Rectangle2D to the specified double values.
void	setRect(Rectangle2D r) Sets this Rectangle2D to be the same as the specified Rectangle2D.
String	toString() Returns the String representation of this Rectangle2D.

Accessing the Device Space dimensions ..

- Useful if you want to scale your window (RasterImage) to fill full screen..
 - **Toolkit** class →
 - part of java.awt
 - Holds data for user/screen space, window elements, etc.
 - **Dimension** class
 - Width and height of a component

[illegible]

Defining components using relative sizing

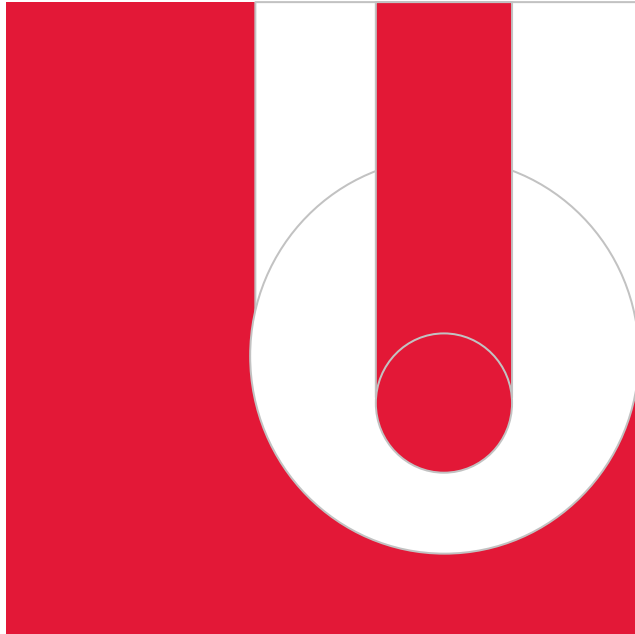
- Example, we want to make YorkU logo:



- What shapes is the logo comprised of?
- Assume:
 - Position: (x,y) or Point2D top left
 - Size: (w,h) ?
 - Could use Rectangle2D (for both)
- Other shapes? (many options)

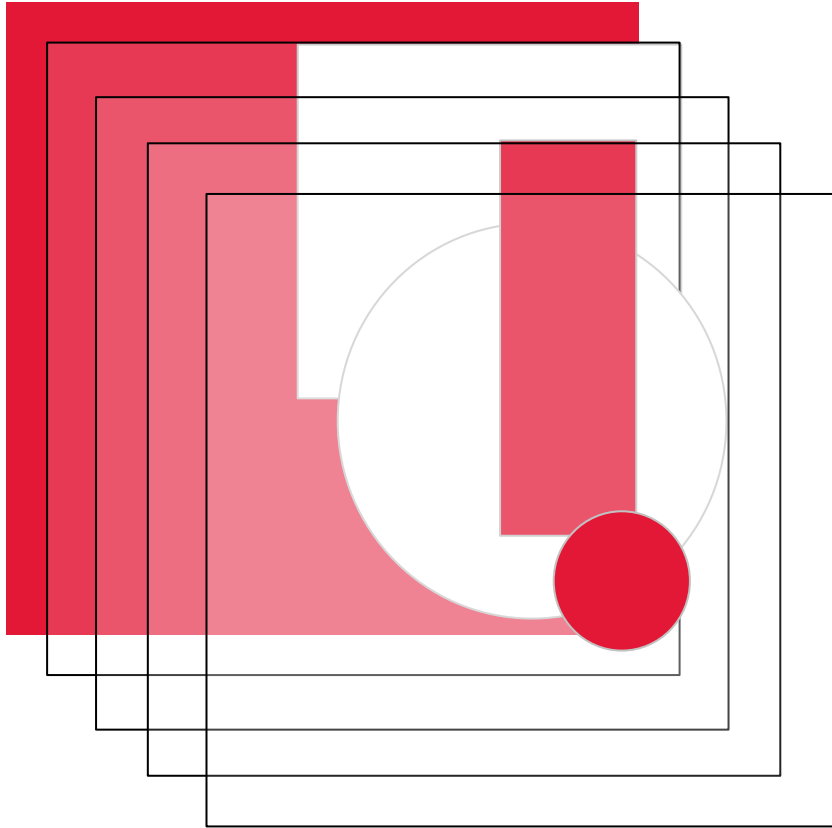


One solution?



Draw order?

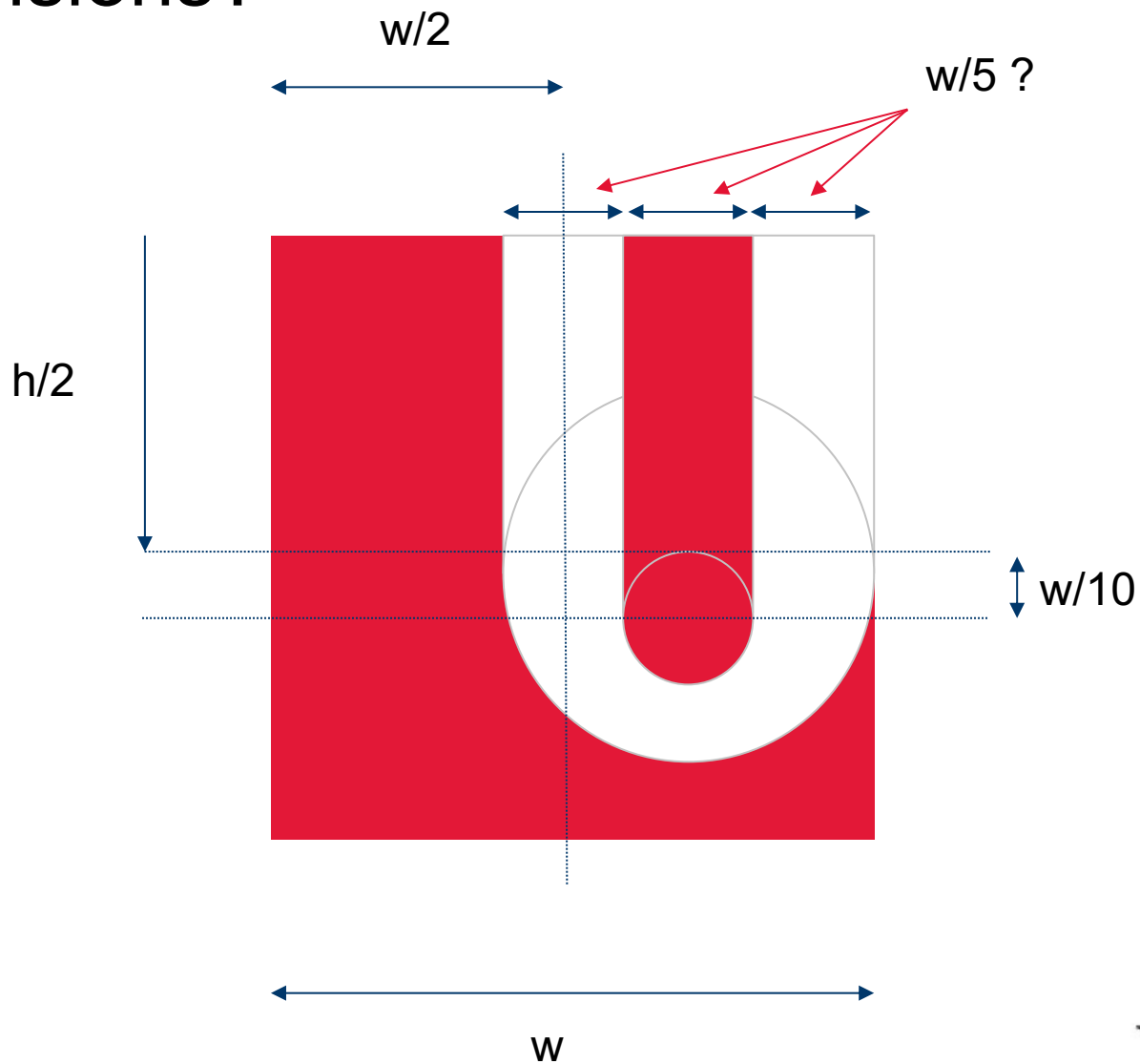
draw bottom layer first



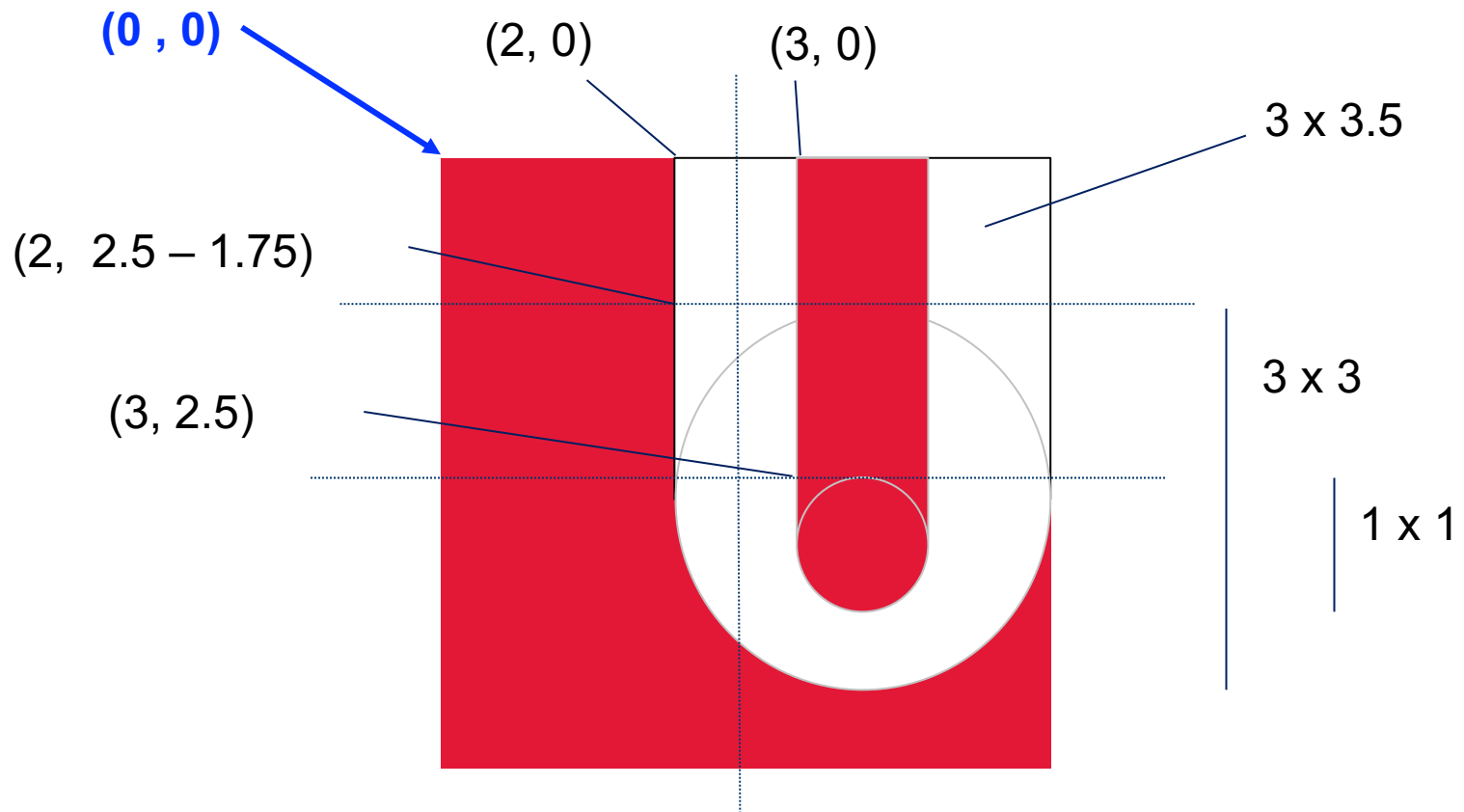
draw top layer last



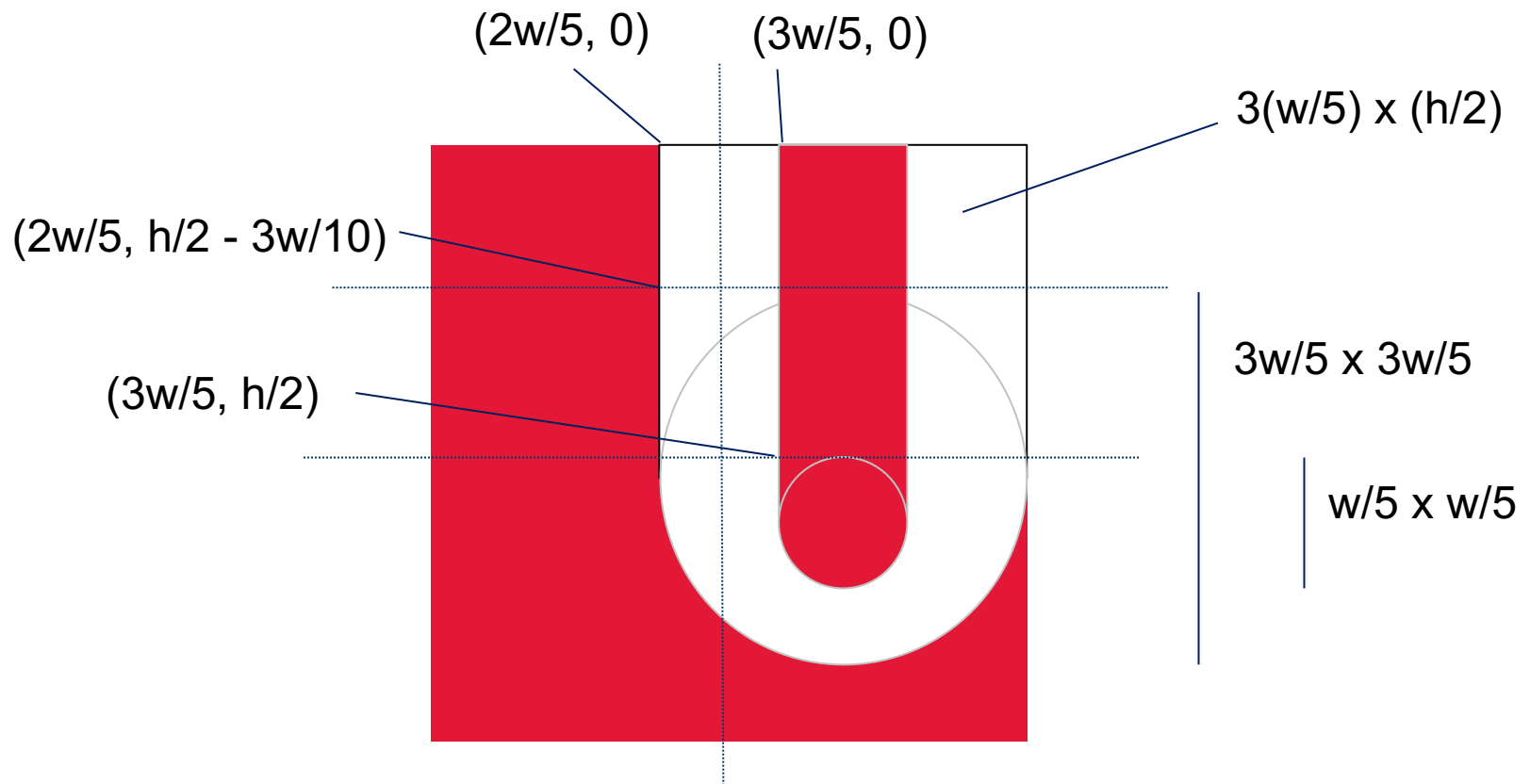
Dimensions?



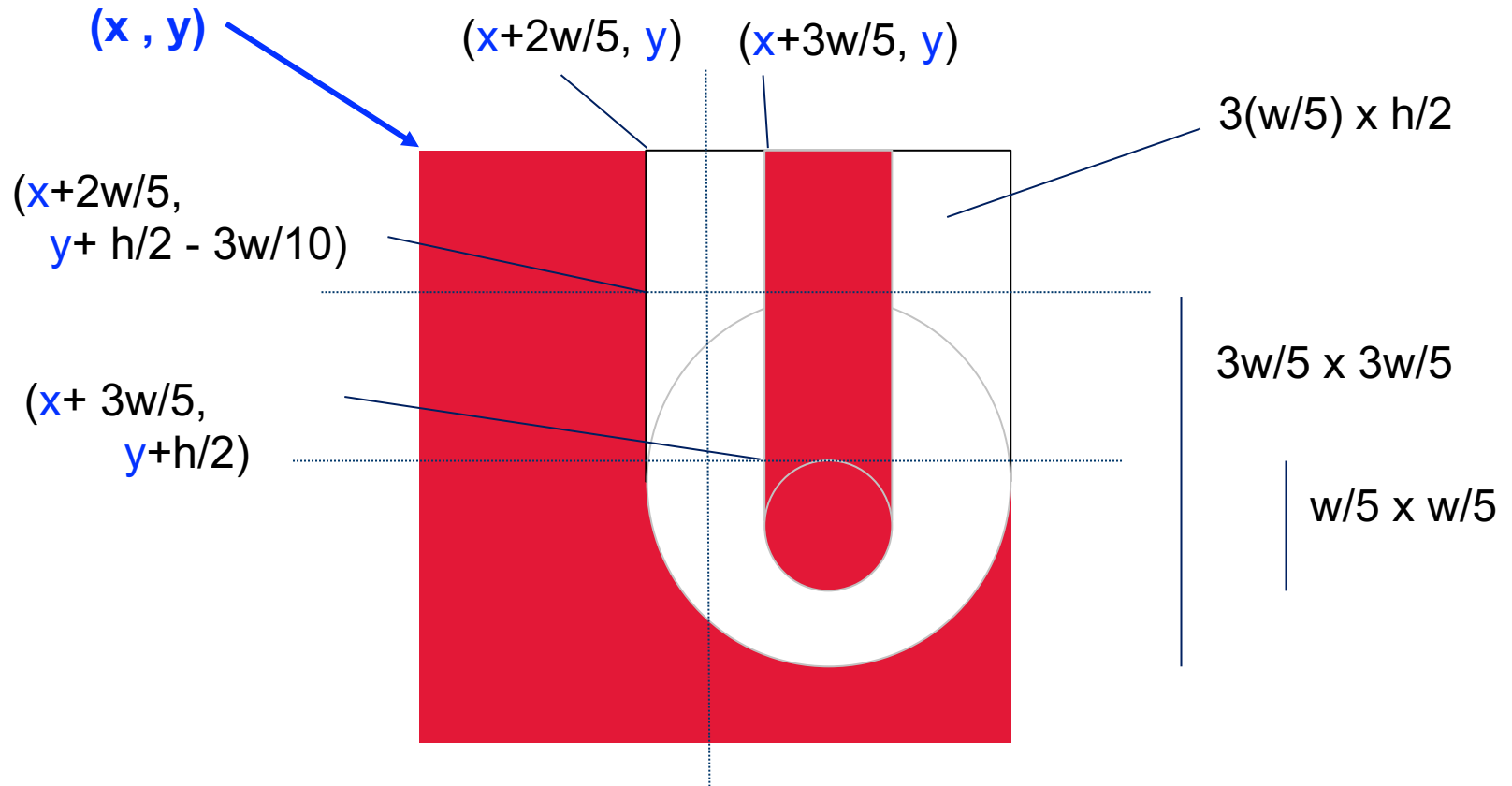
Dimensions? (assume size = 5x5)



Dimensions?



Dimensions?




```
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import imagePackage.RasterImage;
```

```
public class YorkULogo {
```

```
// fields
```

```
private RasterImage img;  
private Graphics2D gfx;  
private Point2D position;  
private Rectangle2D bounds;  
private Rectangle2D backgnd;  
private Rectangle2D whiteRect;  
private Ellipse2D whiteCircle;  
private Rectangle2D redRect;  
private Ellipse2D redCircle;
```

```
// ctors
```

```
public YorkULogo(int x, int y, int w, int h) {
```

```
img = new RasterImage(640,480);
gfx = img.getGraphics2D();
this.position = new Point2D.Double(x,y);
this.bounds = new Rectangle2D.Double(x, y, w, h);
this.backgnd = new Rectangle2D.Double(x, y, w, h);
this.whiteRect = new Rectangle2D.Double(x + 0.4*w , y, 0.6*w, 0.5*h);
this.whiteCircle = new Ellipse2D.Double(x + 0.4*w , y + 0.5*h - 0.3*w,
                                           0.6*w,0.6*w);
this.redRect = new Rectangle2D.Double(x + 0.6*w, y, 0.2*w, 0.5*h);
this.redCircle = new Ellipse2D.Double(x + 0.6*w, y + 0.5*h - 0.1*w ,
                                       0.2*w,0.2*w);
```

}

// ...

Why track things via geometric primitives?

- Can pass them to Graphics2D's draw/fill method

```
// from previous example (gfx is a reference to RasterImage's  
// Graphics2D object  
    this.gfx.draw(this.backgnd);  
    this.gfx.fill(this.backgnd);
```

draw

```
public abstract void draw(Shape s)
```

Strokes the outline of a Shape using the settings of the current Graphics2D context. The rendering attributes applied include the Clip, Transform, Paint, Composite and Stroke attributes.

Parameters:

s - the Shape to be rendered

fill

```
public abstract void fill(Shape s)
```

Fills the interior of a Shape using the settings of the Graphics2D context. The rendering attributes applied include the Clip, Transform, Paint, and Composite.

Parameters:

s - the Shape to be filled

```
// ...
```

```
public void drawLogo() {  
    this.img.show();  
    this.img.setTitle("YorkU");  
    this.gfx.setColor(Color.RED);  
    this.gfx.draw(this.backgnd);  
    this.gfx.fill(this.backgnd);  
  
    this.gfx.setColor(Color.WHITE);  
    this.gfx.draw(this.whiteRect);  
    this.gfx.draw(this.whiteCircle);  
    this.gfx.fill(this.whiteRect);  
    this.gfx.fill(this.whiteCircle);  
  
    this.gfx.setColor(Color.RED);  
    this.gfx.draw(this.redRect);  
    this.gfx.draw(this.redCircle);  
    this.gfx.fill(this.redRect);  
    this.gfx.fill(this.redCircle);  
}  
public static void main(String[] args) {  
    YorkULogo logo = new YorkULogo(0,0,100,100);  
    logo.drawLogo();  
  
    // YorkULogo logo2 = new YorkULogo(150,150,400,400);  
    // logo2.drawLogo();  
}  
}
```