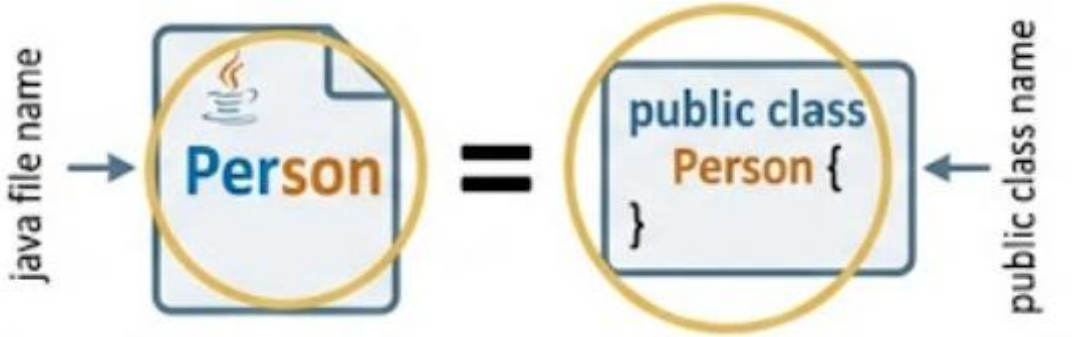
Conceptual Modeling and Programming in GIScience

Lecture 2: Methods and Object Orientation I

Yingjing Huang
yingjing.huang@univie.ac.at

2026S – April 20, 2026

The public class name must exactly match the java file name



What You Will Learn in This Lecture

- Understand what **methods** are and how to create and use them
- Understand why methods **signatures** are key to proper programming
- Learn about **arrays** as a first **data structure**
- Learn about **arrays** as a first data structure
- Learn the basics about **objects and classes** and design your first own class.
- Apply to a practical GIS example: flood analysis using a **Bounding Box**

Primitive Data Types: What about Text?

Recap: ***int***, ***double***, ***boolean***, ***char***, ...

How to store text?

```
String aTextString = "abc";
System.out.println(aTextString);  // abc
```

- String is **not a primitive type** — it's an **object**
- Strings are immutable **sequences** of characters

Casting of Characters

- Java casts **automatically** when safe (**widening**)
- Characters have **numeric values** (ASCII)
- **Parsing:** convert text to numbers

```
// Widening (automatic)
int n = 10;
double d = n;           // 10.0

// Parsing (text → number)
int x = Integer.parseInt("42");
double y = Double.parseDouble("3.14");

// Char ↔ int (ASCII)
char c = (char) 65;     // 'A'
int code = 'A';         // 65
```

American Standard Code for Information Interchange (ASCII)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

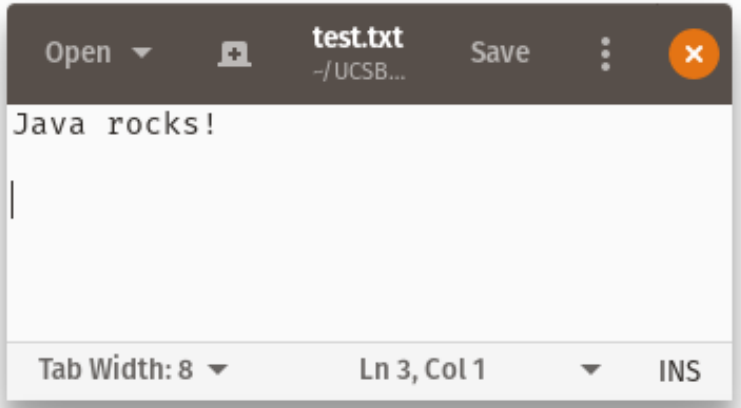
Encoding system for **characters**; now replaced by UTF-8/Unicode

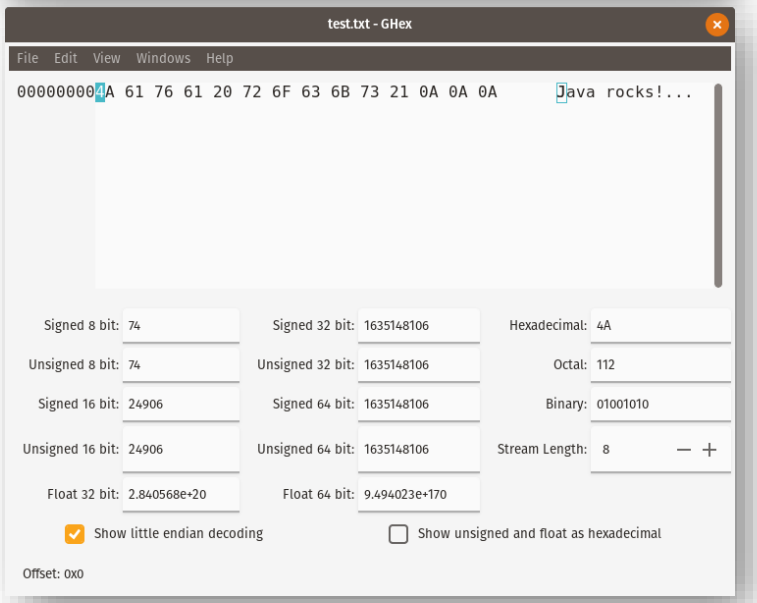
Hex and Special Characters

- **Hex** (base 16): encodes 8 bits using two symbols
- Example:

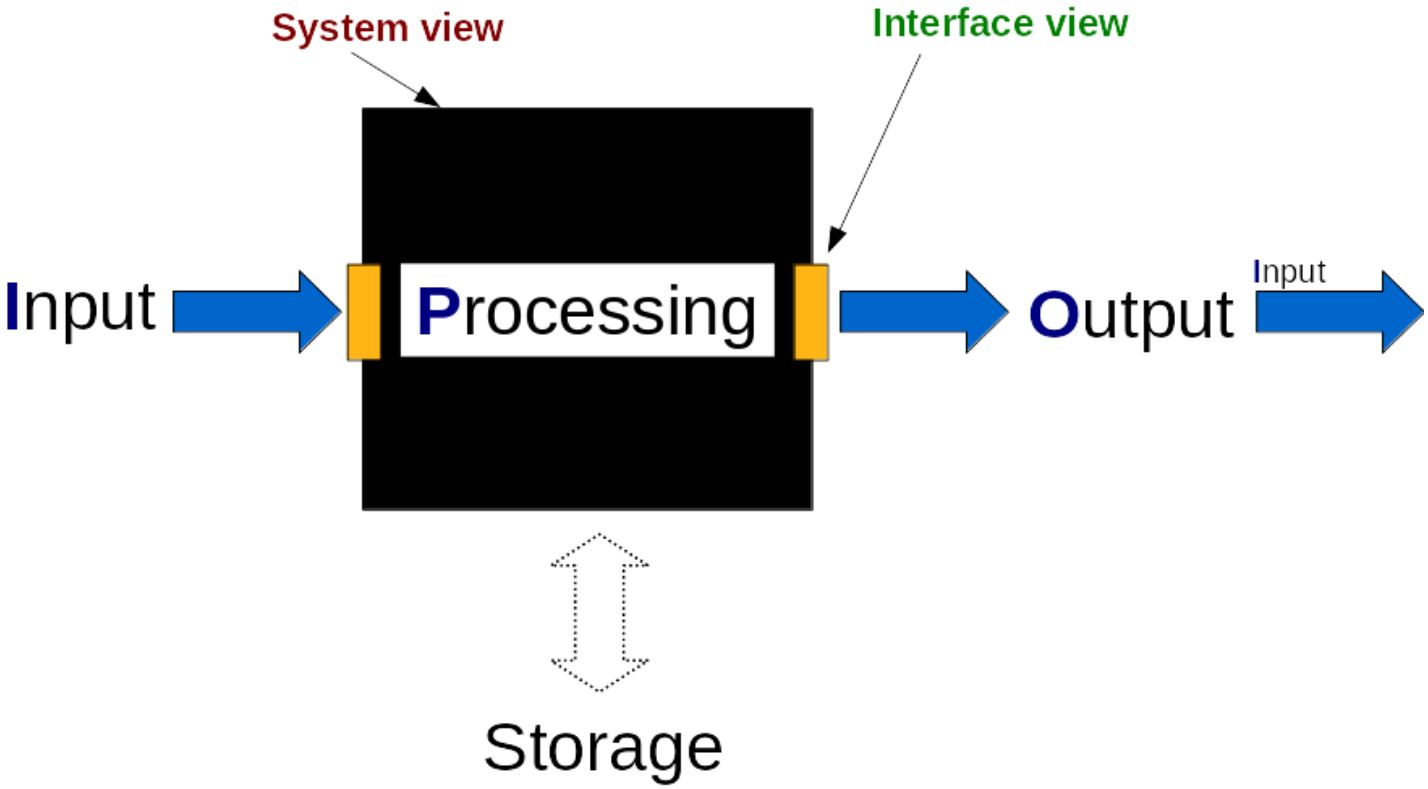
  4A = (4 × 16) + (10 × 1) = **74**

- ASCII includes special characters:
  - **NL** (new line) → \n in Java
  - **TAB** (horizontal tab) → \t in Java

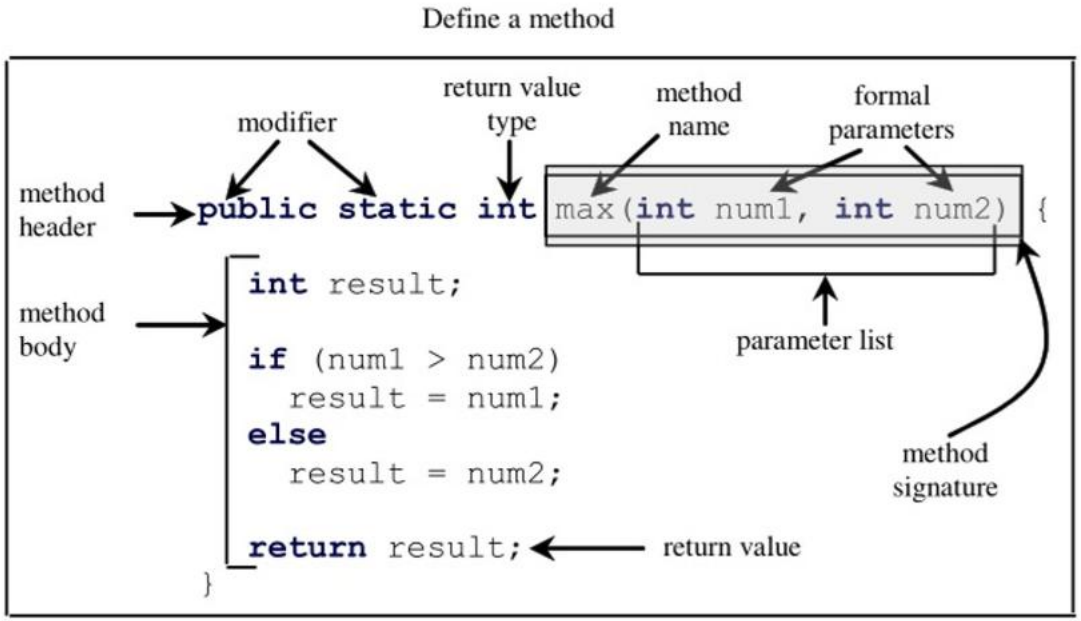




The IPO-Model



- A method: a **reusable** group of instructions with a name
- **Signature** = the interface view (name, inputs, output)
- **body** = the black box (the implementation)
- Methods define the **behavior** of objects



Signatures as Contracts

- A signature is a contract: given these inputs, returns this output
- The caller trusts the signature — doesn't need to read the body
- Clear names make the contract **self-explanatory**
- Teams often agree on signatures first, then write bodies in parallel

modifiers return-types name(parameter-type parameter-name,...) {body}

- Code goes into the **{body}**
- If return type is not **void**, must have a **return** statement.

Examples

```
private static void printAdd2Int(int a , int b) {
    System.out.println(a+b);
} //Printing out is different from returning a value

private static int returnAnInt(int aNumber) {
    return aNumber;
} // returns the number given as parameter
```

Usage example

```
int number = returnAnInt(5);
```

You are already using one method

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

You can **call** other methods from **within** the main method

```
public static void main(String[] args)
      { System.out.println("some text"); }
```

These methods have to be defined **outside** of the main method

```
public static void main(String[] args)
{ your code }
printANumber(int theNumber) …
```

Visibility of Methods and Variables

- **public**, **private**, **protected**, default — control **who can access** your methods and variables
- **Visibility ≠ Scope**
  - Visibility: access control (who can see it)
  - Scope: where a variable exists in memory (e.g., inside a loop)

The following table shows the access to members permitted by each modifier.

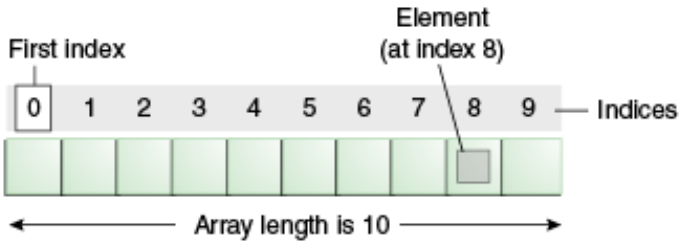
Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Arrays as Data Structure and ADT

- A collection of variables of the same type, accessed by **index**
- Fixed size, declared with a type: **int[]**, **double[]**, ...
- A fundamental **data structure**; behavior defined by **Abstract Data Types** (ADT)
- Think of them as numbered lists for now

```
int[] elevations = new int[5];
elevations[0] = 120;
```



Arrays In Java

Arrays are **objects**,

created with the **new** keyword

- Size is **fixed** at creation
- Indexing starts at **0**

```java
// Declare and create
int[] elevations = new int[3];
elevations[0] = 120;
elevations[1] = 340;
elevations[2] = 85;

// Shorthand
double[] coords = {48.21, 16.37};

// Access
System.out.println(elevations[0]); // 120
System.out.println(coords.length); // 2
```

- **Class** = template (blueprint)

  — defines attributes and behavior

- **Object** = instance

  — a specific individual created from the blueprint

- Classes **declare** attributes, objects **instantiate** them

```
// Class: the blueprint
public class Point { double x, y; }

// Object: a specific instance
Point p = new Point();
```

- **Class** Waterbody
  - **Attributes**
    - water depth
    - Temperature
    - area (**double**)
    - region (an **object** itself)
    - ...
  - **Methods**
    - **isPolluted()** → returns **boolean**
    - **changeState()** → returns **int** (a constant)
    - ...

```
public class Waterbody {
    private double depth;
    private double temperature;
    private double area;
    private Region region;

    public boolean isPolluted() { ... }
    public int changeState() { ... }
}
```

- **Object** Lake1
  - **Attributes**
    - water depth = **50m**
    - polluted = **true**
    - dead zone = **false**
    - region = **Italy**
  - **Methods**
    - **isPolluted()** → **true**
    - **changeState()** → state changes

```
Waterbody lake1 = new Waterbody(50, true, false, "Italy");
lake1.isPolluted();   // true
lake1.changeState();
```

Classes Objects – Syntax

- **Constructor** runs when an object is created; sets initial values
- **this.x** refers to the **member** variable, not the local one
- **Getter & setter** methods control access to member variables
- Class name must match **file name**

```
public class Waterbody {
    private double area;          // member variable

    public Waterbody(double area) {  // constructor
        this.area = area;
    }

    public double getArea() {     // getter
        return this.area;
    }
}

Waterbody wb = new Waterbody(50); // create object
```

Pass by Value: Primitives

- Java passes a **copy** of the value
- Method modifies the copy — original untouched
- "Pass by value"

```
static void addTen(int n) {
    n = n + 10;
}

public static void main(String[] args) {
    int x = 5;
    addTen(x);
    System.out.println(x);   // 5
}
```

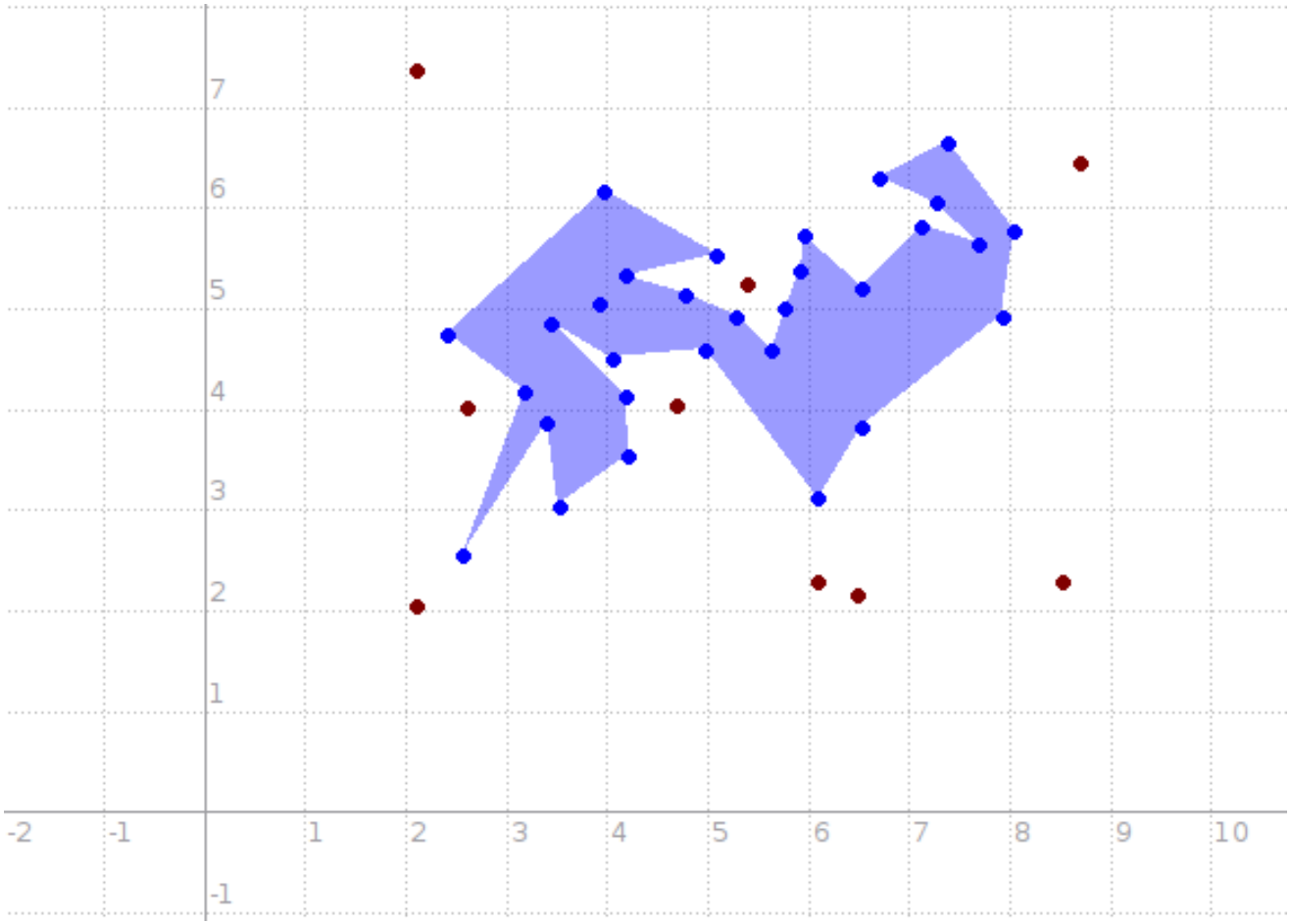
Pass by Value: Objects

- For objects, Java copies the **reference**, not the object
- Modifying the object's state → visible outside
- Reassigning the reference → not visible
- Still "pass by value"

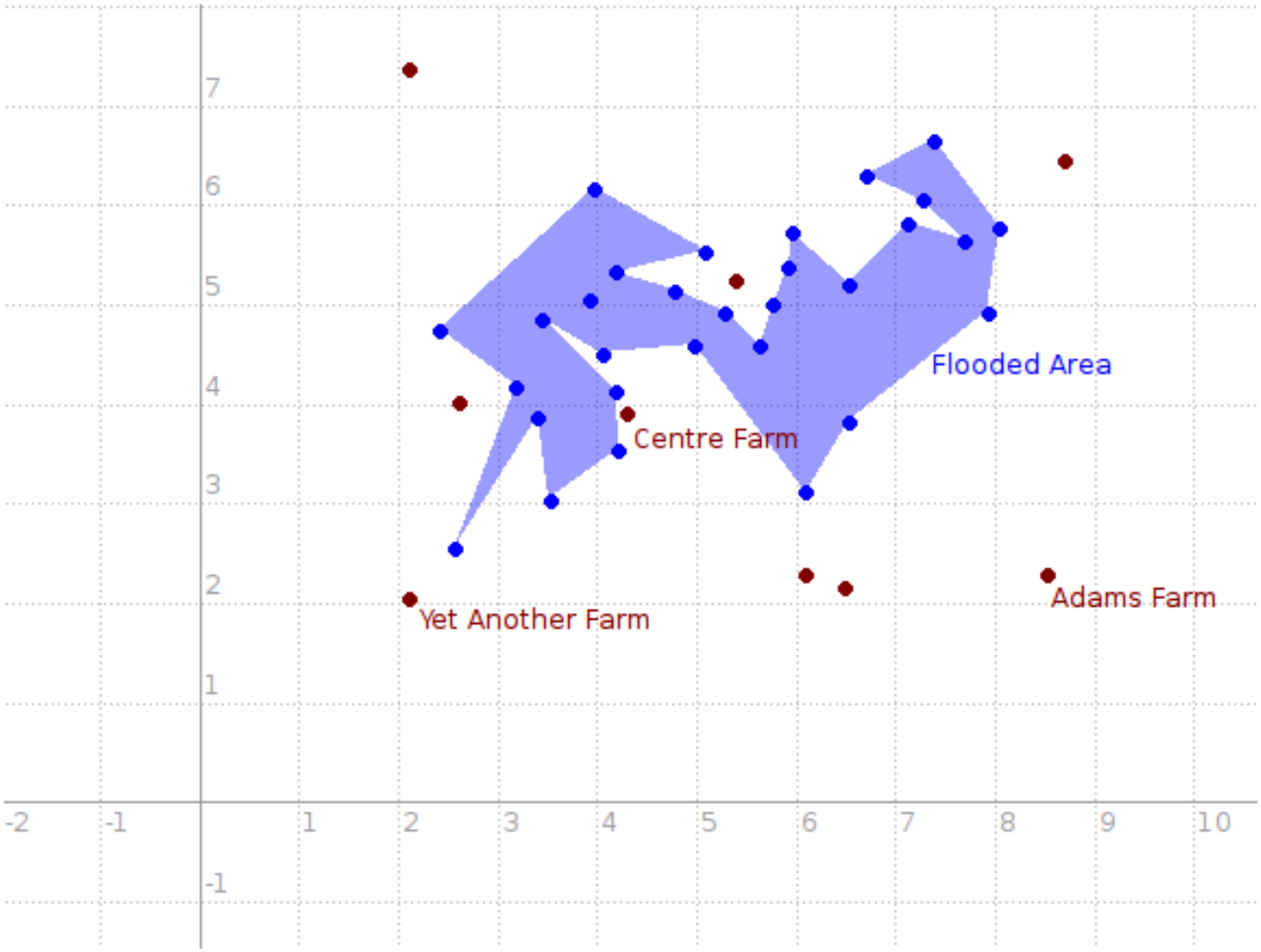
  — just of a reference


```
static void changeFirst(int[] arr) {
      arr[0] = 99;            // modifies the object
  }

static void replace(int[] arr) {
    arr = new int[]{0, 0};  // reassigns the reference
}

public static void main(String[] args) {
    int[] a = {1, 2, 3};
    changeFirst(a);
    System.out.println(a[0]);   // 99

    int[] b = {1, 2, 3};
    replace(b);
    System.out.println(b[0]);   // 1
}
```

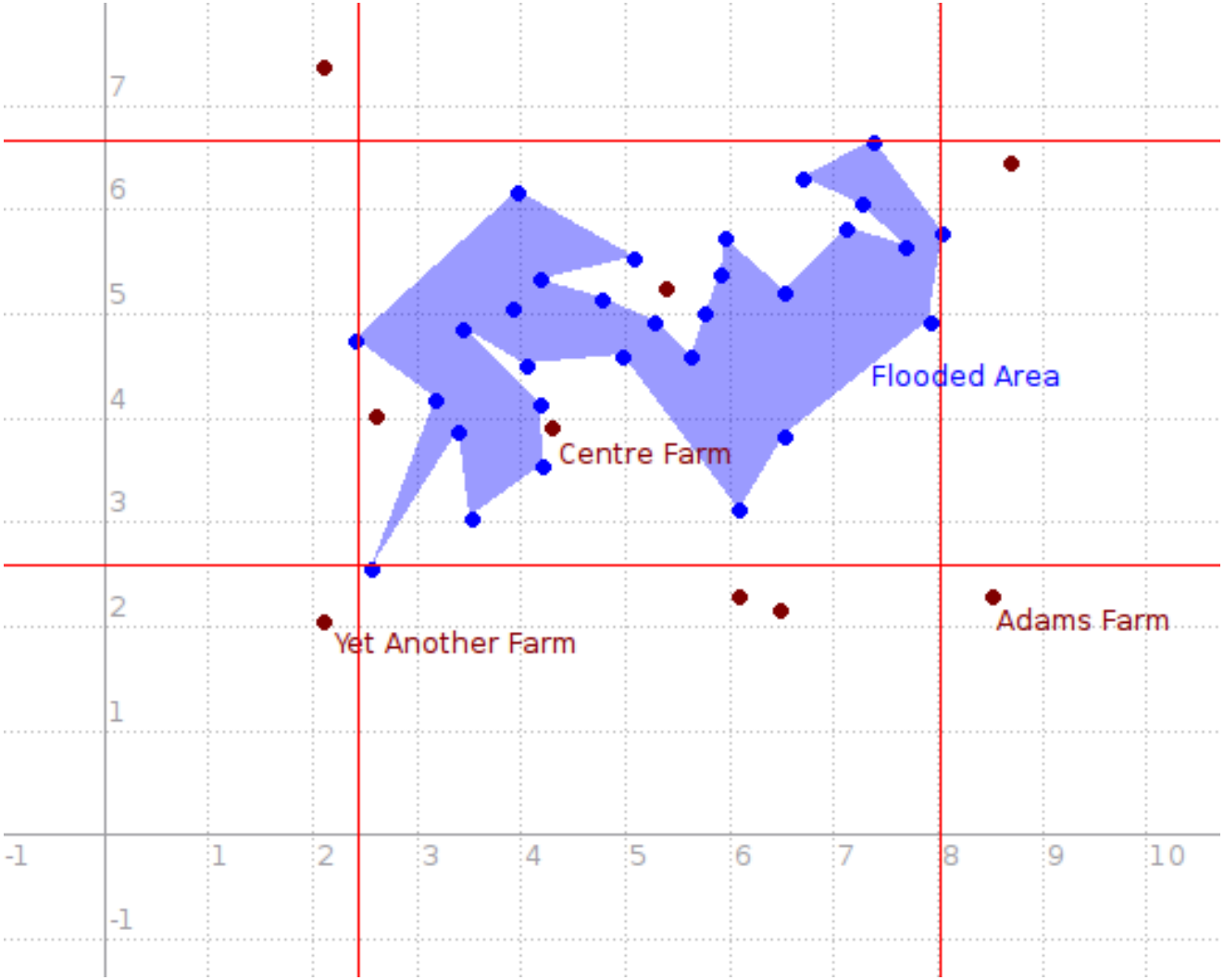
Putting It Together: Bounding Box (Minimum Bounding Rectangle)



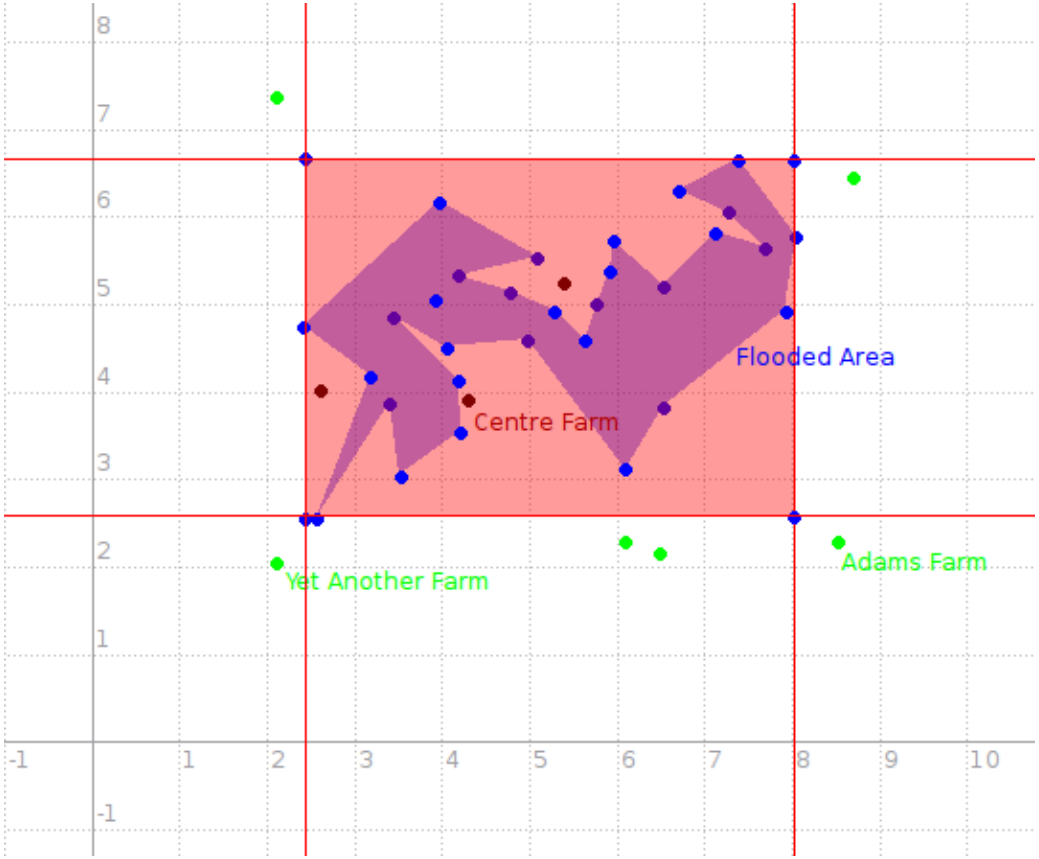
Bounding Box: Flood Scenario



Bounding Box: Drawing the Box



Bounding Box: Lesson Learned



- Computing a bounding box is **simple** — just find min/max x and y
- Use it as a **quick filter**: rule out points that are clearly outside
- Only perform expensive **point-in-polygon** checks on the remaining cases

Arrays in Practice: Farm Example

- Farm data (**name, address, coordinates**) can be stored in arrays
- **Points** and **bounding boxes** can also be represented using arrays

```
Point[] farms = {
  new Point(3.1, 4.2),   // Centre
  new Point(8.0, 5.6),   // Adams
  new Point(1.2, 0.3),   // Yet Another
};

BoundingBox flood =
  new BoundingBox(2, 6, 3, 5);

for (int i = 0; i < farms.length; i++) {
  if (flood.isInside(farms[i])) {
    System.out.println("at risk: " + i);
  }
}
```

Reading

- Describe methods and classes in your own words (short text)

Coding

- Define a **Point** class with a **distanceTo(Point p)** method
- Define a **BoundingBox** class with an **isInside(Point p)** method
- Write a test class: create points and a bounding box, check if points are inside, and compute distances

Submission

- Upload a zip file **[FirstNameLastNameW2.zip]** with the *.java files
- **Deadline**: Sundays at 5:00 PM (Vienna Time) (The day before our Monday lectures).

Lab Time