



Agent and Object Technology Lab  
Dipartimento di Ingegneria dell'Informazione  
Università degli Studi di Parma

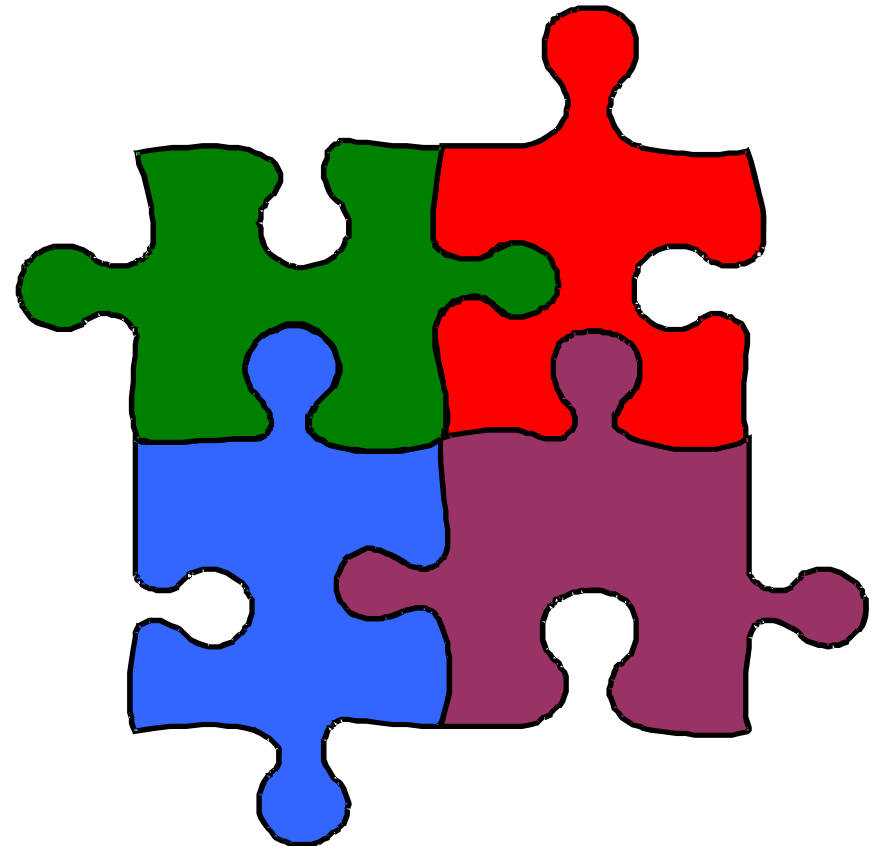


Ingegneria del software A

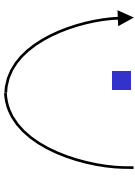
Classi e Oggetti

**Prof. Agostino Poggi**

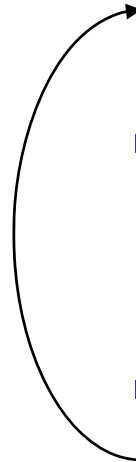
- ◆ Cambia il paradigma della programmazione procedurale
- ◆ Spinta dallo sviluppo delle interfacce grafiche
- ◆ Cerca di massimizzare il riuso di codice
  - Minore time-to-market
  - Minori costi (lavoro)



### ◆ Programmazione procedurale

- Analisi del problema per trovare un algoritmo risolutore
  - Scomposizione dell'algoritmo risolutore in un insieme di algoritmi risolutori più piccoli
- 

### ◆ Programmazione orientata agli oggetti

- Analisi del problema e descrizione dei concetti (oggetti) che ne fanno parte
  - Scomposizione del problema sui singoli oggetti e ricerca di un algoritmo risolutore basato sui servizi forniti da questi oggetti e dalla loro interazione
  - Affinamento dell'algoritmo risolutore in un insieme di algoritmi risolutori più piccoli
- 

- ◆ Sono astrazioni che descrivono:
  - Oggetti fisici
  - Concetti astratti, che fanno parte del problema (o della soluzione)
- ◆ Caratterizzati da:
  - Uno stato
  - Un insieme di servizi che offrono agli altri oggetti
  - Una identità (ad esempio, la locazione in memoria)
- ◆ Esempio, un telefono cellulare
  - Stato: carica della batteria, potenza del segnale, ...
  - Servizi: chiama un numero, rispondi alla chiamata, ...
  - Identità: codice IMEI, posizione fisica, ...

- ◆ Un oggetto può essere descritto tramite le caratteristiche della classe di oggetti di cui fa parte
- ◆ Tutti i telefoni cellulari
  - Hanno un'indicazione del livello della batteria e del segnale ricevuto
  - Consentono di rispondere e di chiamare
- ◆ Spesso si dice *istanza della classe X* anziché *oggetto della classe X*

- ◆ Un oggetto può appartenere a più classi
  - I telefoni cellulari sono anche oggetti fisici
  - Tutti gli oggetti fisici hanno un peso ed una posizione nello spazio
- ◆ Tra classi è possibile stabilire delle relazioni
  - L'insieme degli oggetti definibili dalla classe dei telefoni cellulari è contenuto (sotto-insieme) nell'insieme definibile dalla classe degli oggetti fisici
- ◆ Tra oggetti è possibile stabilire delle relazioni
  - Tutte le batterie hanno un'indicazione del livello di carica
  - Tutti i telefoni cellulari contengono una batteria

- ◆ Sistema procedurale
  - Insieme di procedure che si chiamano l'una con l'altra
- ◆ Sistema ad oggetti
  - Insieme di oggetti che richiedono l'esecuzione di servizi ad altri oggetti del sistema
- ◆ La complessità diminuisce tanto più quanto
  - Lo stato dei singoli oggetti è semplice
  - L'insieme dei servizi offerti dai singoli oggetti è coeso (ma generale)

- ◆ Nei linguaggi basati sulle classi (come Java e C++)
  - Gli oggetti vengono descritti solo tramite le loro classi
  - Gli oggetti vengono creati partendo dalle loro classi
  - Un oggetto ha una sola classe origine
- ◆ Per realizzare un sistema ad oggetti bisogna
  - Definire le classi
    - Stato, mediante un insieme di attributi
    - Servizi, mediante un insieme di metodi
    - Relazioni con altre classi, mediante un insieme di attributi e metodi
  - Creare gli oggetti del sistema e farli interagire attraverso l'invio di messaggi



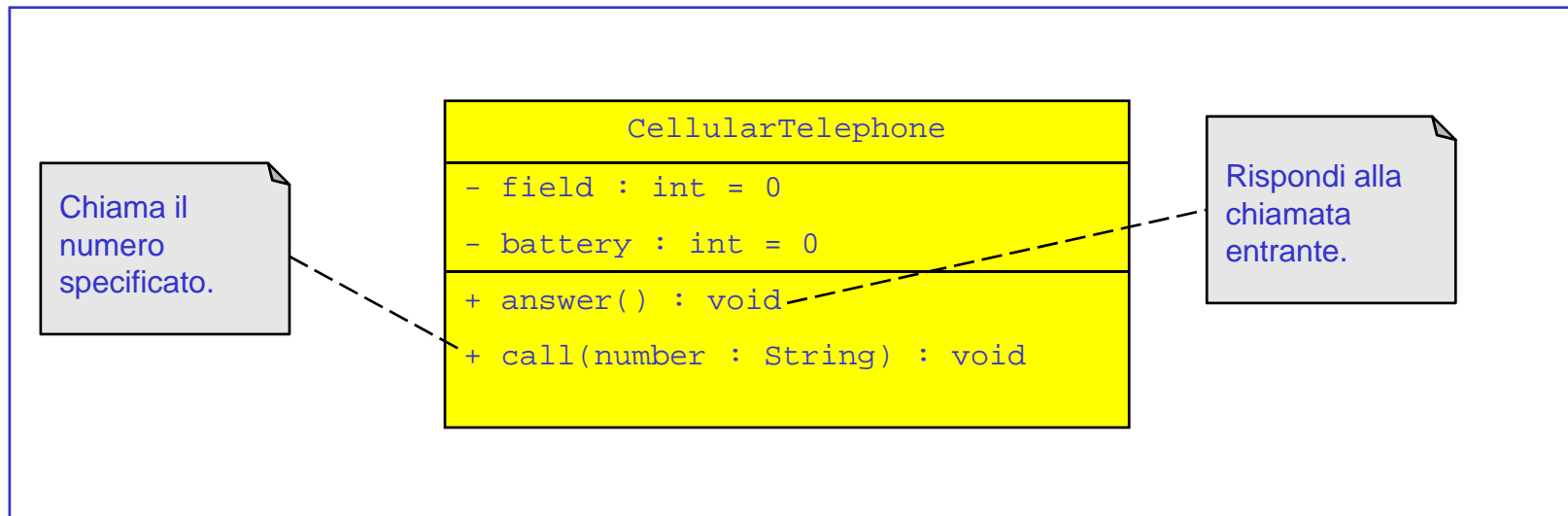
```
public class CellularTelephone {  
  
    private int field = 0;  
    private int battery = 0;  
  
    public void answer() {  
        ...  
    }  
  
    public void call(String number) {  
        ...  
    }  
}
```

attributi che descrivono lo stato

metodi che descrivono i servizi

1/CellularTelephone.java

- ◆ Un diagramma che contiene classi e relazioni tra classi si chiama **class diagram**



- ◆ Realizzare un sistema che calcola l'area e il perimetro di tre rettangoli
- ◆ Primo passo è descrivere gli oggetti che fanno parte del problema
  - Classe dei rettangoli
- ◆ Secondo passo è creare gli oggetti ed inviare i messaggi necessari

```
public class Rectangle {  
  
    private int width  = 0;  
    private int height = 0;  
  
    public int area() {  
        return this.width * this.height;  
    }  
  
    public int perimeter() {  
        return 2 * this.width + 2 * this.height;  
    }  
  
}
```

Rectangle

- width : int = 0

- height : int = 0

+ area() : int

+ perimeter() : int

2/Rectangle.java

- ◆ Tutti i file sorgenti
  - Contengono una sola classe
  - Hanno il nome della classe
  - Hanno l'estensione “.java”
- ◆ La compilazione del file *Rectangle.java* avviene tramite il comando:  

```
javac Rectangle.java
```
- ◆ Se la compilazione ha successo, viene generato il bytecode *Rectangle.class*

- ◆ Gli oggetti vengono creati partendo dalla loro classe di appartenenza

- Si crea un oggetto rettangolo tramite il codice:

```
Rectangle obj = new Rectangle();
```

- Si invia un messaggio all'oggetto `obj` tramite il codice:

```
obj.area();
```

- ◆ La variabile `obj` è detta **object reference** ed è una sorta di puntatore all'oggetto rettangolo

- ◆ C'è un grosso problema nel codice precedente
  - L'oggetto creato dal costruttore della classe `Rectangle` ha larghezza e altezza nulla
- ◆ Esistono due soluzioni:
  - Introdurre dei metodi di inizializzazione
  - Introdurre un costruttore specifico

- ◆ Spesso si introducono dei metodi che consentono di manipolare direttamente lo stato di un oggetto

```
public void setWidth(final int w) { this.width = w; }  
public int getWidth() { return this.width; }
```

- ◆ L'uso dei metodi **getter** e **setter** dovrebbe essere minimizzato
  - Solo l'oggetto è responsabile di come il suo stato viene memorizzato
  - Tutti i servizi utili che si basano sullo stato dell'oggetto dovrebbero essere offerti dall'oggetto stesso



- ◆ Quindi la soluzione con i metodi di inizializzazione è basata su due passi

- L'introdurre dei metodi di inizializzazione

```
public int setWidth(final int w) {  
    this.width = w; }  
public int setHeight(final int h) {  
    this.height = h; }
```

- La creazione e l'inizializzazione dell'oggetto

```
Rectangle r = new Rectangle();  
r.setWidth(3);  
r.setHeight(4);
```

- ♦ La soluzione precedente ha vari problemi
  - I metodi `setWidth` e `setHeight` sono stati aggiunti solo allo scopo di inizializzare l'oggetto
  - Il programmatore potrebbe dimenticarsi di invocarli dopo la costruzione dell'oggetto
- ♦ È necessario che il linguaggio consenta di tenere unite la **costruzione** e l'**inizializzazione** degli oggetti

- ◆ Principio **instantiation is initialization**:
  - *La creazione di un oggetto coincide con la sua inizializzazione ad uno stato iniziale*
- ◆ Per soddisfare questo principio è necessario prevedere che venga chiamata una procedura d'inizializzazione, al momento della creazione
- ◆ Questo è possibile creando un costruttore specifico

```
public Rectangle(final int w, final int h) {  
    this.width = w; this.height = h;  
}
```

- ◆ L'istruzione `new Rectangle(4, 5)` ha l'effetto di
  - Creare l'oggetto in memoria
  - Chiamare il costruttore dell'oggetto
- ◆ Una classe può dichiarare più costruttori differenziati, parleremo di **overloading** in base al numero e al tipo dei parametri
  - L'overloading si applica anche per i metodi comuni
  - Nella prima riga di un costruttore, si può usare `this(...)` per chiamare un altro costruttore
- ◆ Costruttore di default, senza parametri
  - Creato da Java se (e solo se) non è definito nessun costruttore

```
public class Rectangle {
```

```
    private int width  = 0;  
    private int height = 0;
```

```
    public Rectangle(final int w, final int h) {  
        this.width  = w; this.height = h;  
    }
```

```
    public int area() {  
        return this.width * this.height;  
    }
```

```
    public int perimeter() {  
        return 2 * this.width + 2 * this.height;  
    }
```

```
}
```

Rectangle

- width : int = 0

- height : int = 0

+ Rectangle(w : int, h : int)

+ area() : int

+ perimeter() : int

3/Rectangle.java

- ◆ Per avviare un sistema è necessario fornire alla JVM una procedura per iniziare l'esecuzione di un programma
- ◆ Questa procedura (metodo) può essere definita in ogni classe e si chiama **main**
- ◆ Spesso si definisce una classe con il solo scopo di contenere questo metodo

```
public class Main {  
    public static void main(final String[] args) {  
        // Creazione dei tre rettangoli  
        Rectangle r1 = new Rectangle(5, 7);  
        Rectangle r2 = new Rectangle(4, 8);  
        Rectangle r3 = new Rectangle(10, 10);  
  
        // Calcolo delle tre aree  
        int a1 = r1.area();  
        int a2 = r2.area();  
        int a3 = r3.area();  
  
        // Stampa delle tre aree  
        System.out.println("Area del rettangolo 1: " + a1);  
        System.out.println("Area del rettangolo 2: " + a2);  
        System.out.println("Area del rettangolo 3: " + a3);  
    }  
}
```

3/Main.java

- ♦ Una classe descrive le caratteristiche che avranno i suoi oggetti (metodi e attributi di istanza), ma può anche descrivere delle caratteristiche associabili alla classe (metodi e attributi di classe)
- ♦ La parola chiave **static** indica questo tipo di metodi e attributi
  - Non sono associati ad una particolare istanza
  - Sono disponibili al di fuori di un oggetto della classe
  - Non possono accedere direttamente ai metodi e campi di istanza
  - Sono accessibili tramite il nome della classe (e.g., `Math.PI` e `Math.sqrt(5.2)`)



- ♦ L'esecuzione del file *Main.class* avviene tramite il comando:

```
java Main
```

- ♦ Il generale il comando java può ricevere più parametri
  - Il primo è il nome della classe di cui eseguire il metodo `main`
  - I successivi sono i parametri da passare a questo metodo attraverso l'array `args`

- ♦ La creazione di un oggetto richiede memoria per memorizzare lo stato dell'oggetto
- ♦ Quando un oggetto non serve più è necessario liberare questa memoria
- ♦ La JVM offre un **garbage collector** per gestire in modo automatico la restituzione della memoria
- ♦ Quando un oggetto non serve più, il garbage collector lo distrugge

- ◆ Vantaggi

- Non è possibile dimenticare di liberare la memoria
- Non è possibile liberare della memoria che dovrà essere utilizzata in seguito

- ◆ Svantaggi

- Il garbage collector decide autonomamente quando liberare la memoria
- Liberare e compattare la memoria richiede del calcolo

### ◆ Dichiarazione

```
int[] k;  
String[] names;
```

A differenza del C, non si può  
specificare la dimensione

### ◆ Creazione

```
k = new int[3];  
names = new String[50];
```

### ◆ Lunghezza

```
k.length
```

### ◆ Eccezione

```
ArrayIndexOutOfBoundsException
```

L'indice è fuori dai limiti

### ◆ Multi-dimensionali

```
double[][] dbl = new double[3][3];
```

### ◆ Inizializzazione

```
int[][] id3 = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
```

- ◆ Classe e literal

```
String greeting = "Hello world!";
```

- ◆ Lunghezza

```
int len = greeting.length();
```

- ◆ Concatenazione

```
string1.concat(string2);  
"My name is ".concat("Rumplestiltskin");  
"Hello," + " world" + "!";
```

- ◆ Comparazione

```
string1.equals(string2);  
string1.compareTo(string2);  
string1.startsWith(string2);
```

- ◆ I caratteri sono rappresentati da una codifica numerica
- ◆ Esistono diverse codifiche raggruppate in tre standard
  - ASCII
  - ISO 8859
  - UNICODE

- ♦ ASCII (American Standard Code for Information Interchange) utilizza 7 bit (estesa a 8 bit)
  
- ♦ L'ASCII codifica:
  - I caratteri alfanumerici (lettere maiuscole e minuscole e numeri), compreso lo spazio
  
  - I simboli (punteggiatura, @, #, ...)
  
  - Alcuni caratteri di controllo che non rappresentano simboli visualizzabili (TAB, LF, CR, BELL, etc.)

CTRL	Dec.	Esa.	Car.	Cod.	Dec.	Esa.	Car.	Dec.	Esa.	Car.	Dec.	Esa.	Car.
^@	0	00		NUL	32	20	!	64	40	@	96	60	'
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	"	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	\$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	'	71	47	G	103	67	g
^H	8	08		BS	40	28	(	72	48	H	104	68	h
^I	9	09		HT	41	29	)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B		ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	
^]	29	1D		GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^~	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	¸



# Tabella ASCII estesa (Cp437)

Dec.	Esa.	Car.	Dec.	Esa.	Car.	Dec.	Esa.	Car.	Dec.	Esa.	Car.
128	80	Ç	160	A0	á	192	C0	L	224	E0	α
129	81	ü	161	A1	í	193	C1	⊥	225	E1	β
130	82	ë	162	A2	ó	194	C2	T	226	E2	Γ
131	83	â	163	A3	ô	195	C3	└	227	E3	Π
132	84	ä	164	A4	û	196	C4	—	228	E4	Σ
133	85	à	165	A5	ñ	197	C5	+	229	E5	σ
134	86	å	166	A6	ä	198	C6	⋈	230	E6	μ
135	87	ç	167	A7	ø	199	C7	⋈	231	E7	Υ
136	88	ê	168	A8	¿	200	C8	⋈	232	E8	ϕ
137	89	ë	169	A9	¬	201	C9	⋈	233	E9	Θ
138	8A	è	170	AA	½	202	CA	⋈	234	EA	Ω
139	8B	ì	171	AB	¼	203	CB	⋈	235	EB	δ
140	8C	î	172	AC	¼	204	CC	⋈	236	EC	∞
141	8D	ï	173	AD	•	205	CD	=	237	ED	Φ
142	8E	Ĥ	174	AE	«	206	CE	⋈	238	EE	ε
143	8F	Ĥ	175	AF	»	207	CF	⋈	239	EF	∩
144	90	Ĥ	176	B0	▤	208	D0	⋈	240	F0	≡
145	91	æ	177	B1	▤	209	D1	⋈	241	F1	±
146	92	Œ	178	B2	▤	210	D2	⋈	242	F2	≥
147	93	ô	179	B3	▤	211	D3	⋈	243	F3	≤
148	94	ö	180	B4	▤	212	D4	⋈	244	F4	┌
149	95	ò	181	B5	▤	213	D5	⋈	245	F5	└
150	96	û	182	B6	▤	214	D6	⋈	246	F6	÷
151	97	ù	183	B7	▤	215	D7	⋈	247	F7	≈
152	98	ÿ	184	B8	▤	216	D8	⋈	248	F8	◊
153	99	ÿ	185	B9	▤	217	D9	⋈	249	F9	•
154	9A	Ü	186	BA	▤	218	DA	⋈	250	FA	·
155	9B	¢	187	BB	▤	219	DB	▤	251	FB	√
156	9C	£	188	BC	▤	220	DC	▤	252	FC	≡
157	9D	¥	189	BD	▤	221	DD	▤	253	FD	²
158	9E	℔	190	BE	▤	222	DE	▤	254	FE	³
159	9F	ƒ	191	BF	▤	223	DF	▤	255	FF	■

- ◆ Code page 437 per Pc (Dos) in Nord America
  - Caratteri accentati
  - Caratteri per grafici
- ◆ Possibile combinare testo in inglese e francese (anche se in Francia si usava Cp850), ma non in inglese e greco (Cp737)
- ◆ Iso 8859 – estensioni standard per ASCII ad 8 bit
  - Iso 8859-1 (o Latin1) – Lingue dell'Europa Occidentale
  - Iso 8859-5 – Alfabeto cirillico
  - Iso 8859-7 – Alfabeto greco
  - Iso 8859-15 – Latin1 con simbolo euro (€)

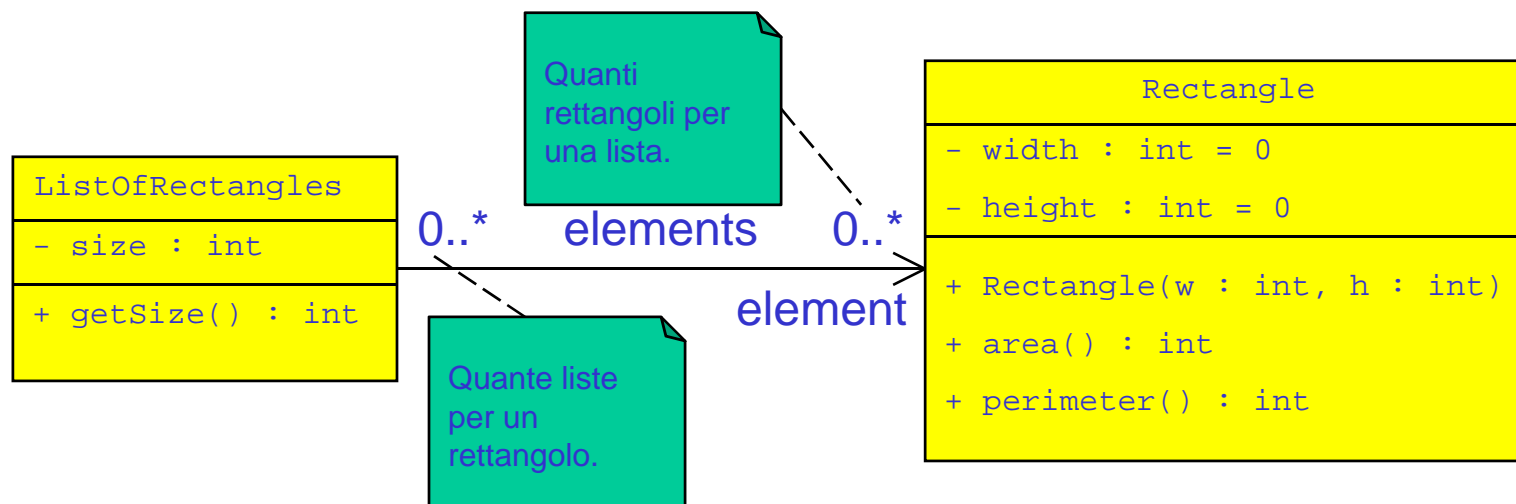
- ◆ Associa un preciso code-point a ciascun simbolo
- ◆ Possibile rappresentare milioni di simboli (32 bit)
- ◆ Per ora comprende più di 30 sistemi di scrittura
  - Proposte per geroglifici e caratteri cuneiformi
  - Proposta per Klingon (da Star Trek... rifiutata!)
- ◆ Primi 256 code-points = Latin1
- ◆ Unicode definisce diverse codifiche
  - UTF-8 – codifica con base ad 8-bit ma lunghezza variabile (1-4 byte), massima compatibilità con ASCII
  - UTF-16 – base a 16-bit, lunghezza variabile
  - UTF-32 – codifica a 32-bit, lunghezza fissa

```
String str = "abcd\u5B66\uD800\uDF30";  
int charCount = str.length();  
int codePointCount = str.codePointCount(0, charCount);  
int utf8Count = str.getBytes("UTF-8").length;
```

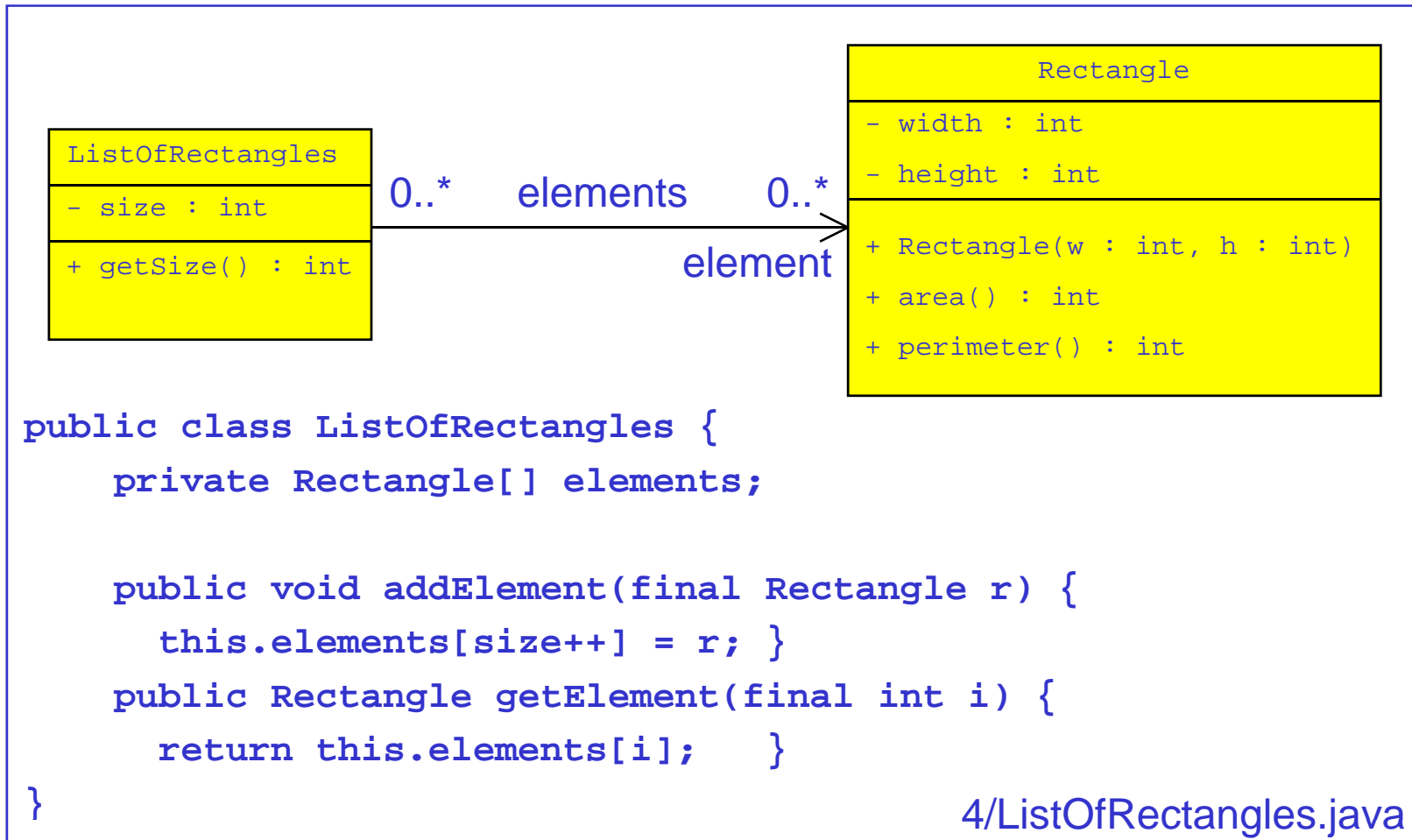
<b>String</b>	abcd学p
<b>Char Count</b>	7
<b>Character Count</b>	6
<b>UTF-8 Byte Count</b>	11

- ◆ Realizzare un programma che stampi l'area di 10 rettangoli contenuti in una lista
  
- ◆ Classi necessarie
  - Classe dei rettangoli
  
  - Classe delle liste di rettangoli

- ◆ C'è una relazione tra un oggetto lista di rettangoli e un insieme di rettangoli
- ◆ Una relazione tra oggetti genera un'associazione tra le classi



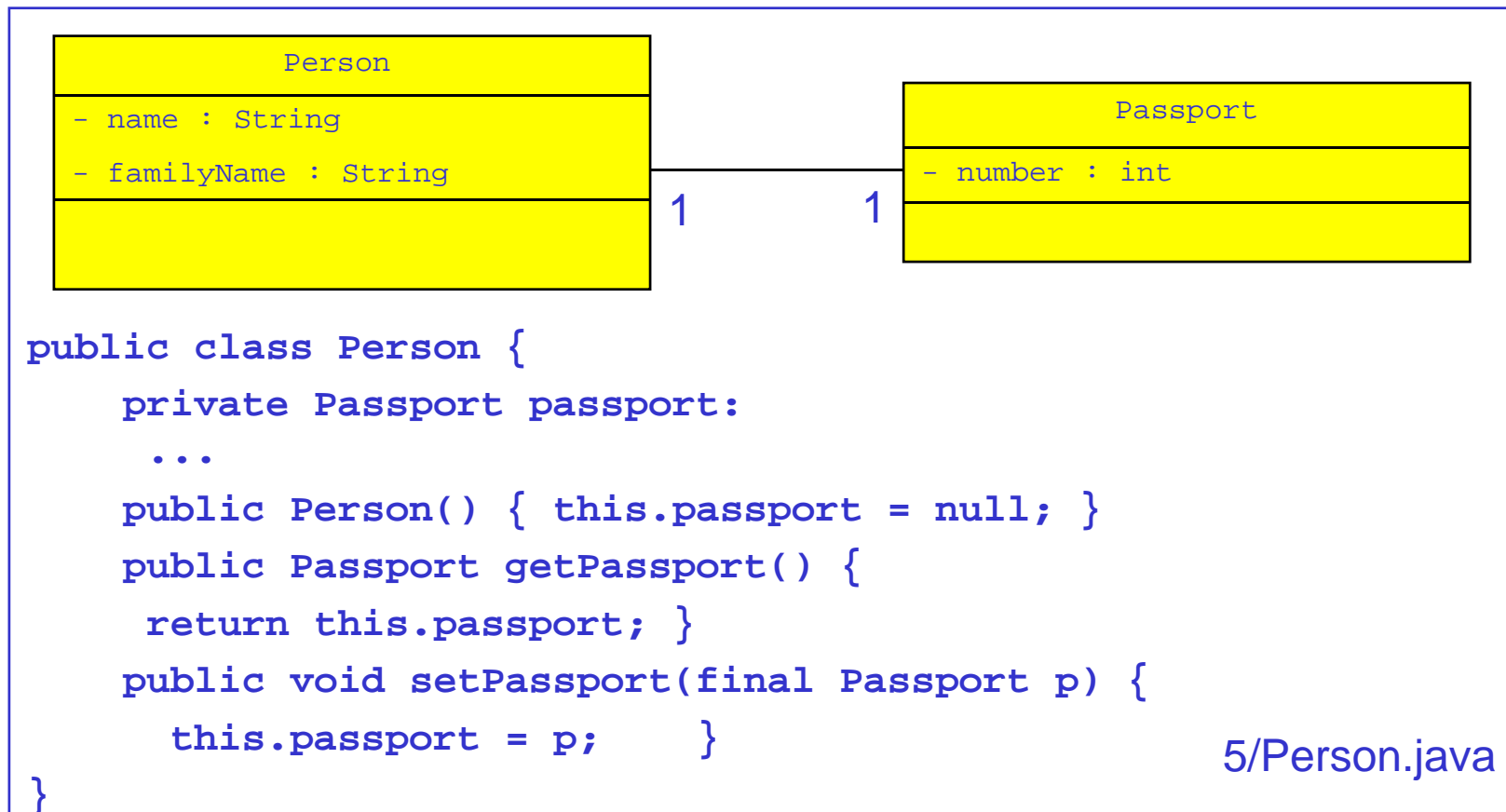
- ◆ Un'associazione ha un nome che la descrive
- ◆ Ognuno dei lati
  - Ha un nome
  - Ha una cardinalità
  - Può essere navigabile o no
- ◆ Un'associazione è implementata tramite del codice Java



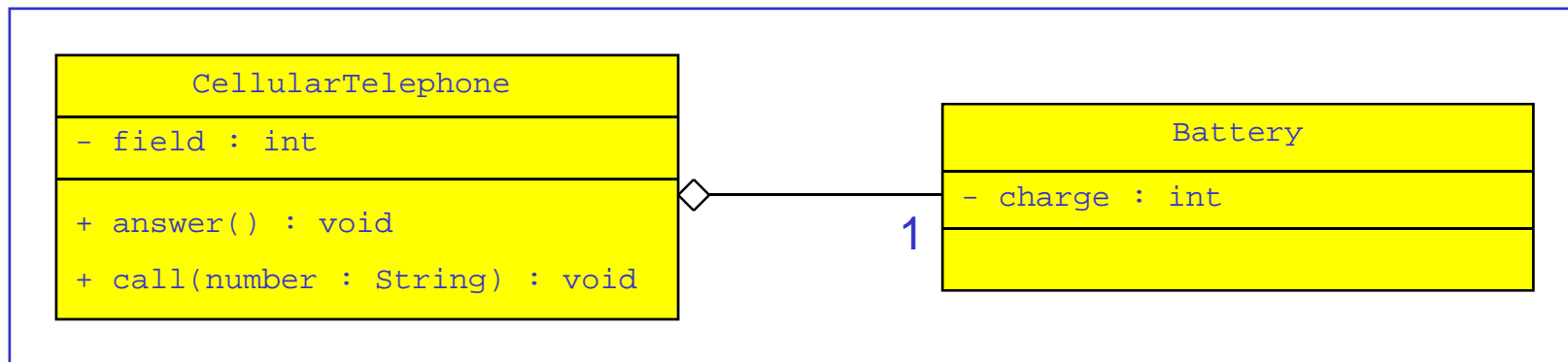


- ◆ È importante notare che
  - Solo i lati navigabili implementano l'associazione
  - È necessario controllare il vincolo sulla cardinalità (e.g., nell'esempio della lista di rettangoli tramite i metodi `addElement` e `removeElement`)
  - Ale relazioni possono essere anche implementate tramite i cosiddetti **oggetti aggregati** (e.g., `ArrayList` e `TreeSet`)

- ◆ Un'associazione particolarmente importante è l'associazione 1 a 1

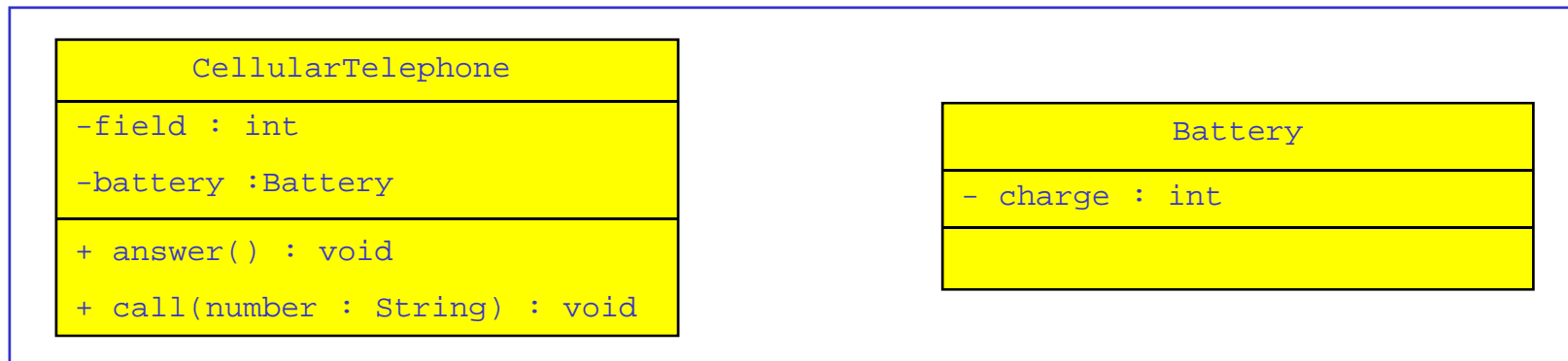
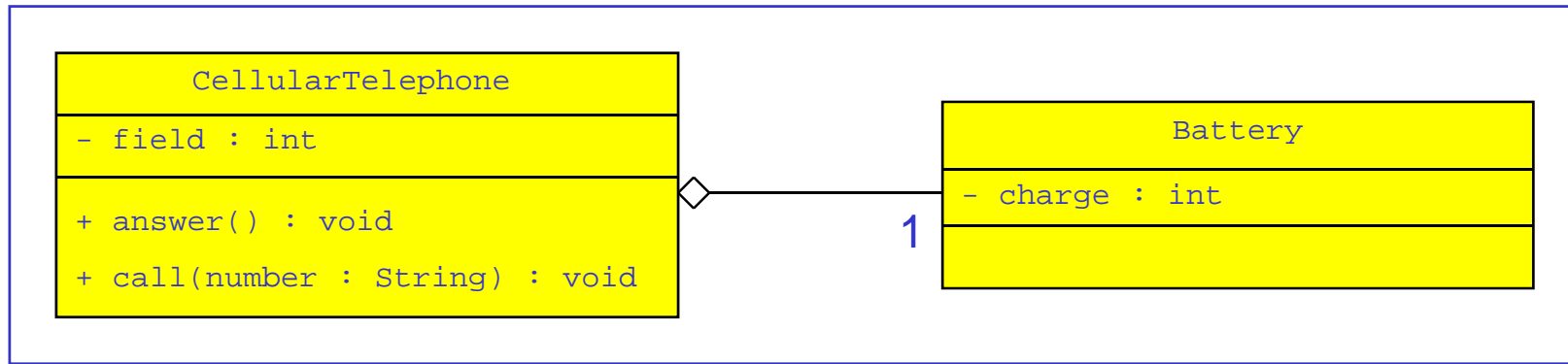


- ◆ Se un'associazione esprime un contenimento fisico, o una relazione del tipo, **part-of**, o una relazione del tipo, **has-a**, allora si parla di **relazione di contenimento**
  - Può essere composizione o aggregazione
- ◆ In Java non c'è nessuna vera differenza tra contenimento ed un più generica associazione



- ◆ Idealmente, un oggetto rappresenta un'unità di codice, che altri oggetti possono usare
- ◆ Un oggetto può contenere un certo numero di oggetti di tipo diverso, per realizzare le funzionalità desiderate
- ◆ Questo comporta un elevato grado di flessibilità
  - Gli oggetti membri sono di solito nascosti
  - Inaccessibili ai programmatori che usano l'oggetto
  - Possono essere cambiati senza disturbare il codice esterno

- ◆ Associazioni e attributi del modello generano codice
- ◆ Le associazioni vengono usate per collegare tra loro le classi del problema (o della soluzione)
- ◆ Gli attributi sono in genere di tipo **primitivo** (e.g., int o float) oppure classi offerte dalle librerie di base di Java (e.g., String o ArrayList)
- ◆ Spesso si modellano come attributi anche delle classi del problema che non si vogliono includere esplicitamente nel problema (o nella soluzione)



- ◆ Realizzare un programma che ordini una lista di 10 rettangoli in base alla loro area.
- ◆ Classi necessarie
  - Classe dei rettangoli
  - Classe delle liste di rettangoli
- ◆ L'ordinamento verrà effettuato richiedendo il servizio “ordinati” ad un oggetto di classe lista di rettangoli

```
public class ListOfRectangles {
    ...
    public void sort() {
        int size = this.elements.size();

        for(int i = 0; i < size; i++)
            for(int j = i + 1; j < size; j++) {
                Rectangle left  = this.elements[i];
                Rectangle right = this.elements[j];

                if(left.area() > right.area()) swap(i, j);
            }
    }
    private void swap(final int i, final int j) {
        Rectangle t = this.elements[i];
        this.elements[i] = this.elements[j];
        this.elements[j] = t;
    }
}
```

6/ListOfRectangles.java



- ♦ Il metodo `swap ( )` è privato
  - Non è un servizio che l'oggetto offre ad altri oggetti
  - Può essere chiamato solo all'interno dell'oggetto stesso
- ♦ I metodi privati sono assimilabili a procedure o funzioni nell'approccio procedurale
  - Il loro scopo è scomporre gli algoritmi implementati dai metodi pubblici

- ◆ Esistono vari algoritmi di ordinamento applicabili ad una lista, il metodo `sort()` utilizza l'algoritmo comunemente indicato dal nome *selection sort*
- ◆ Il problema è che abbiamo legato l'algoritmo alla struttura dati
- ◆ Quando si vuole implementare un algoritmo che può essere **sostituito**, conviene utilizzare una classe anziché un metodo

```
public class SelectionSorter {
    public void sort(ListOfRectangles list)
    {
        for (int i = 0; i < list.getSize(); i++)
            for (int j = i + 1; j < list.getSize(); j++) {
                Rectangle left  = list.getElement(i);
                Rectangle right = list.getElement(j);

                if (left.area() > right.area())
                    swap(list, i, j);
            }
    }

    private void swap(final ListOfRectangles list, final int i, final int j)
    {
        Rectangle t = list.getElement(i);

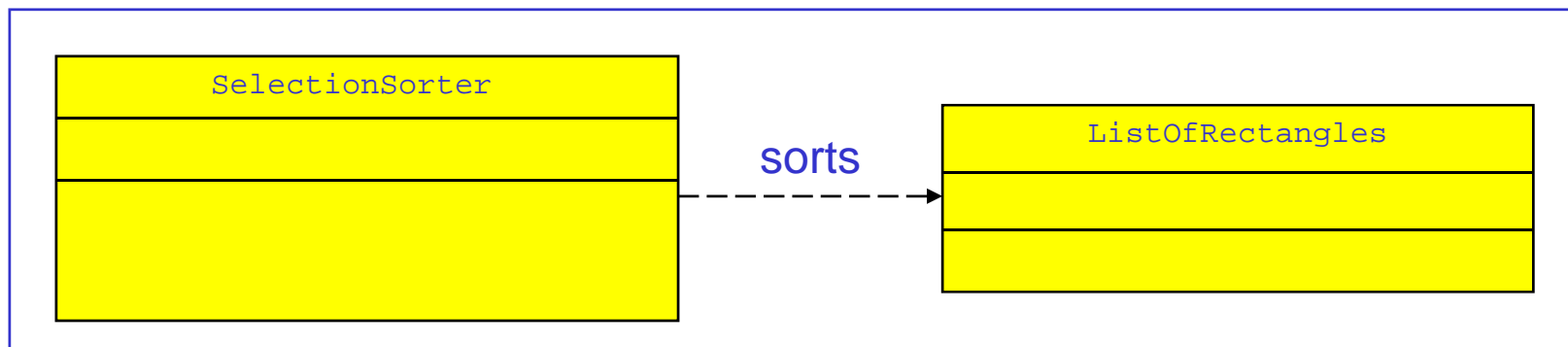
        list.setElement(i, list.getElement(j));
        list.setElement(j, t);
    }
}
```

7/SelectionSorter.java

- ◆ Per poter implementare il SelectionSorter dobbiamo dare accesso agli elementi della lista attraverso dei metodi pubblici (i.e., `setElement` e `getElement`)
- ◆ È importante che l'accesso agli elementi interni non sveli come la lista memorizza i propri elementi
  - (i.e., un array, una lista concatenata, ...)
- ◆ Questo offre la possibilità di
  - Cambiare la memorizzazione interna
  - Aumentare la riusabilità perché SelectionSorter è applicabile indipendentemente da come la lista è memorizzata

- ◆ Principio di **information hiding**
  - Bisogna nascondere i dettagli implementativi di un oggetto
- ◆ **Incapsulamento**
  - Gli utilizzatori sono insensibili ai cambiamenti nell'implementazione
  - L'interfaccia pubblicata (i.e., l'insieme dei metodi pubblici) può essere tenuta stabile, ma senza influire su possibili modifiche nella loro implementazione
- ◆ L'oggetto è gestibile come una black box in cui:
  - L'attenzione è ai servizi che offre e non come essi sono implementati
  - La soluzione é scalabile per problemi complessi

- ♦ La classe `SelectionSorter` prevede che
  - Esista nel sistema una classe `ListOfRectangles`
  - La classe `ListOfRectangles` abbia i metodi `getElement()`, `setElement()` e `getSize()`
- ♦ La classe `SelectionSorter` **dipende** dalla classe `ListOfRectangles`



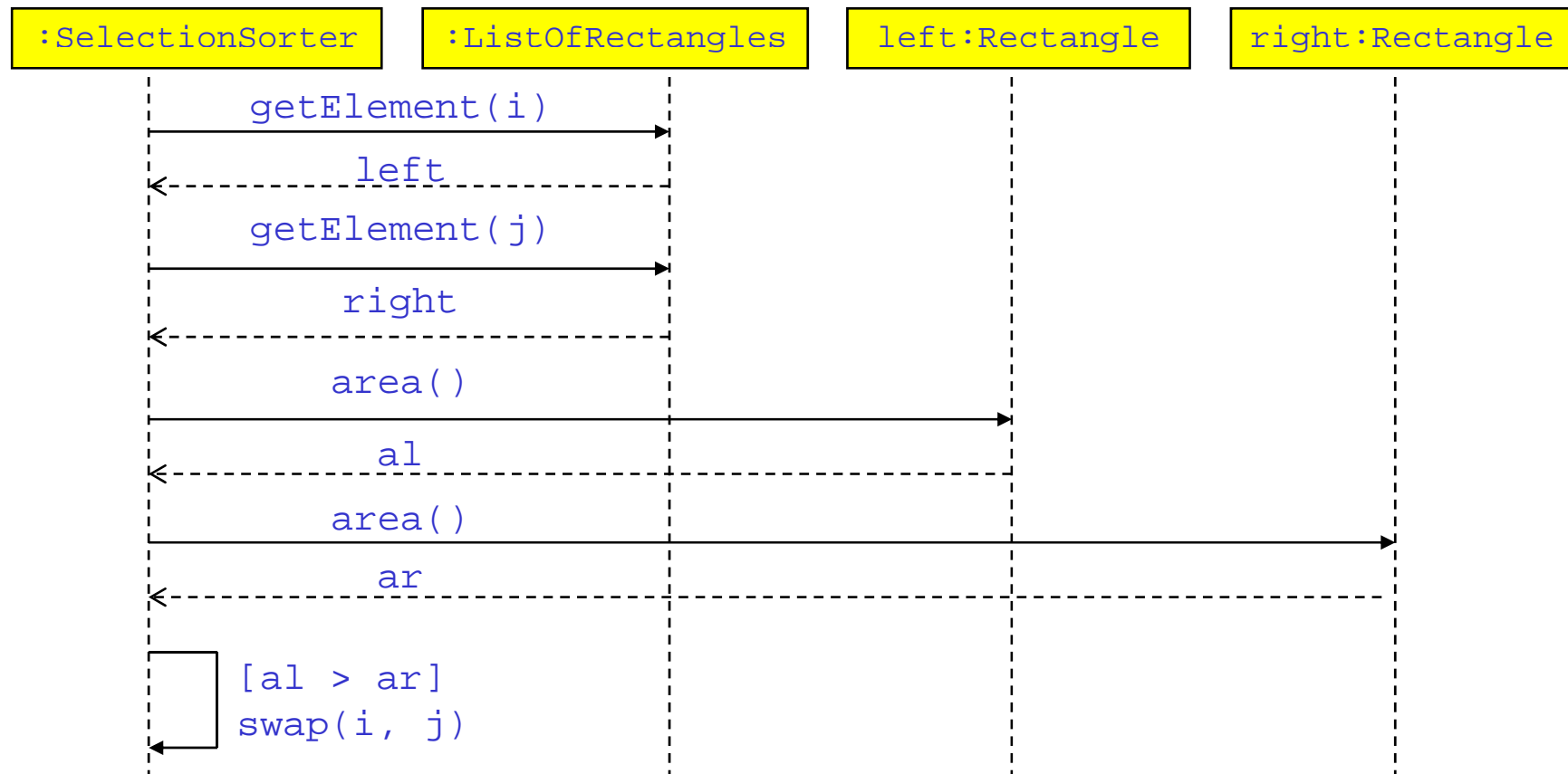
- ♦ Una classe può dipendere da servizi di un'altra classe, ma non deve avere nessuna struttura interna che dipende da quest'ultima
- ♦ La forma più comune di dipendenza è nel passaggio di parametri
  - Un metodo di una classe ha come tipo di un argomento un'altra classe (con cui non è associata)
- ♦ Dipendenze e associazioni limitano la riusabilità perché **accoppiano** le classi

- ◆ UML mette a disposizione due tipi principali di diagrammi, detti **interaction diagram**, per descrivere in modo grafico le interazioni tra gli oggetti di un programma
- ◆ **Sequence diagram**
  - Mette in evidenza la successione degli eventi nel tempo
- ◆ **Communication diagram**
  - Mette in evidenza le relazioni tra oggetti

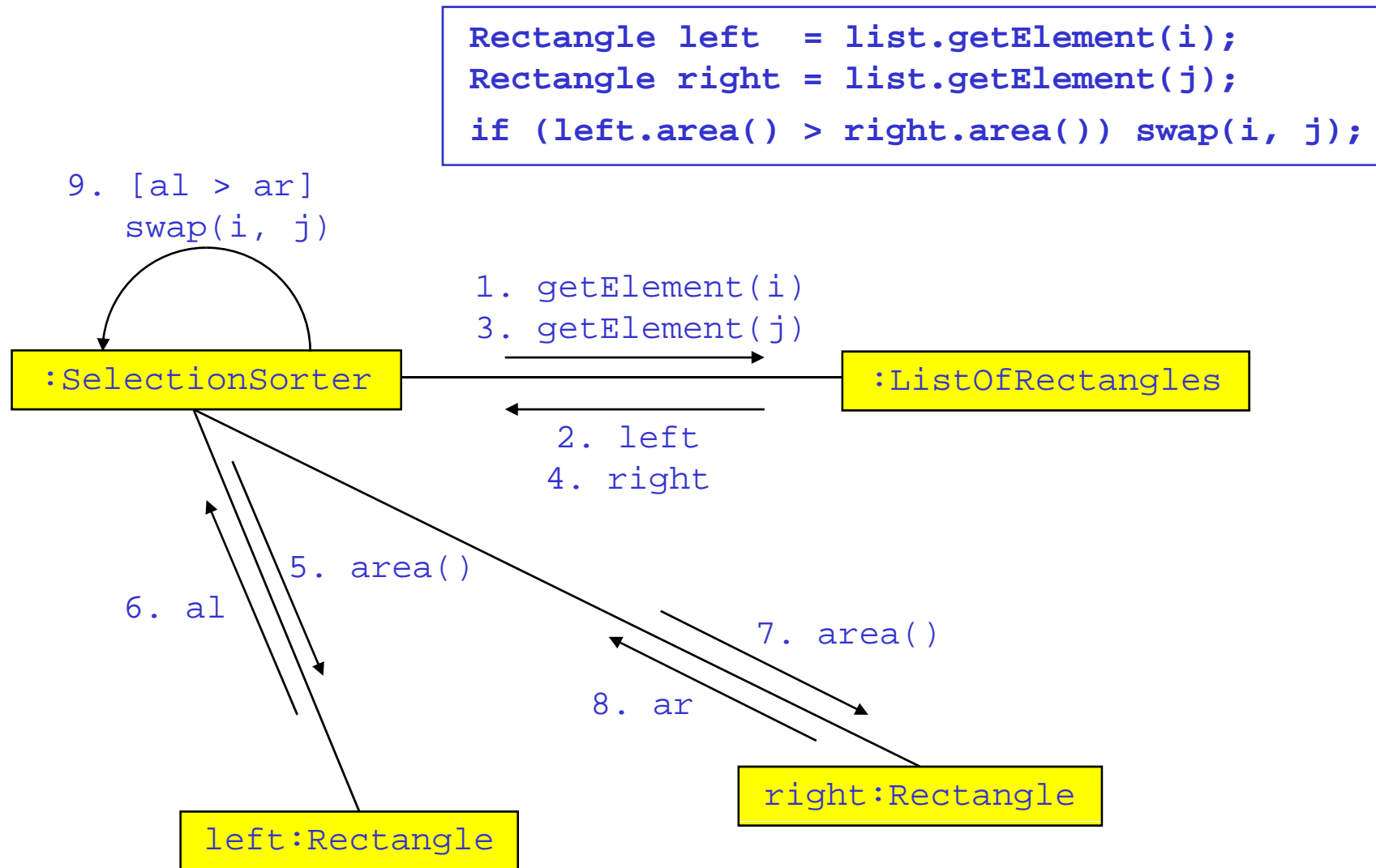


## Sequence diagram

```
Rectangle left  = list.getElement(i);
Rectangle right = list.getElement(j);
if (left.area() > right.area()) swap(i, j);
```



## Communication diagram



- ◆ Vantaggio
  - Rappresentazione grafica utile per comprendere meglio interazioni complesse
- ◆ Svantaggi
  - Non sufficientemente espressivi
  - Potenzialmente ambigui
  - Più ridondanti di uno pseudo-codice
- ◆ Utili
  - In fase di analisi e di progettazione ad alto livello
  - Come documentazione