Azzolini Riccardo

Paradigmi di comunicazione

1 Blackboard

Una comunicazione di tipo **blackboard** è simile a quella di tipo buffer, ma:

- la lettura dei dati è *non distruttiva*, cioè i dati letti rimangono disponibili, e possono quindi essere letti più volte;
- i messaggi lasciati sulla blackboard vengono preservati finché non sono sovrascritti, cancellati o invalidati;
- la scrittura è solitamente non bloccante.

I principali metodi di una blackboard sono:

- `write`, per scrivere un messaggio;
- `clear`, per cancellare i messaggi;
- `read`, che legge un messaggio, bloccando il chiamante finché non ci sono dati validi sulla blackboard;
- `dataAvailable`, che indica se la blackboard ha dati disponibili, e può essere usato dal lettore per evitare di bloccarsi.

```
public class Blackboard<Data> {
    private Data message;
    private boolean valid;

    public Blackboard() {
        valid = false;
    }

    public Blackboard(Data initial) {
        message = initial;
        valid = true;
    }

    public synchronized void write(Data message) {
        this.message = message;
        valid = true;
```

```
        notifyAll();
    }

    public synchronized void clear() {
        valid = false;
    }

    public synchronized Data read() throws InterruptedException {
        while (!valid) {
            wait();
        }
        return message;
    }

    public boolean dataAvailable() {
        return valid;
    }
}
```

Osservazione:

- Sono disponibili due costruttori: uno senza argomenti, che crea una blackboard "vuota", e uno che permette invece di specificare un messaggio iniziale (quindi la blackboard conterrà subito dei dati validi).
- I metodi `write` e `clear` non sono bloccanti: il messaggio corrente viene sovrascritto, che sia stato letto o meno.
- Invece, il metodo `read` è bloccante: se non è già presente un messaggio valido, si aspetta che ne venga scritto uno.

1.1 Esempio: canale digitale terrestre

Un canale digitale terrestre propone ogni giorno una sequenza di programmi. Un utente vuole vedere le informazioni su cosa stiano trasmettendo in qualsiasi momento. Per farlo, si sintonizza e legge le informazioni sul programma che stanno trasmettendo in un determinato istante.

I messaggi scritti sulla blackboard saranno istanze della classe `Programma`:

```
public class Programma {
    private final String nome;
    private final String oraInizio;
    private final int durataInMin;
```

```
    public Programma(String nome, String oraInizio, int durataInMin) {
        this.nome = nome;
        this.oraInizio = oraInizio;
        this.durataInMin = durataInMin;
    }

    public String toString() {
        return nome + " (inizia alle " + oraInizio + ")";
    }

    public int durataInMin() {
        return durataInMin;
    }
}
```

Ogni thread `Utente` ha un riferimento alla blackboard, dalla quale effettua una lettura (eventualmente bloccandosi) per "guardare" il programma corrente:

```
public class Utente extends Thread {
    private Blackboard<Programma> blackboard;

    public Utente(Blackboard<Programma> blackboard) {
        setName("Utente " + getName());
        this.blackboard = blackboard;
    }

    public void run() {
        try {
            System.out.println(getName() + ": attendo programma");
            Programma msg = blackboard.read();
            System.out.println(getName() + ": sto guardando " + msg);
        } catch (InterruptedException e) {}
    }
}
```

Il thread `CanaleTv` ha un array di programmi da trasmettere. Per ciascuno di essi, lo scrive sulla blackboard, e poi fa una `sleep` per simulare la durata del programma:

```
public class CanaleTv extends Thread {
    private final String nome;
    private Blackboard<Programma> blackboard;
    private Programma[] programmazione;

    public CanaleTv(
        String nome,
```

```
        Blackboard<Programma> blackboard,
        Programma[] programmazione
    ) {
        this.nome = nome;
        this.blackboard = blackboard;
        this.programmazione = programmazione;
    }

    public void run() {
        for (Programma programma : programmazione) {
            System.out.println(nome + " trasmette " + programma);
            blackboard.write(programma);
            try {
                Thread.sleep(programma.durataInMin());
            } catch (InterruptedException e) { return; }
        }
    }
}
```

Infine, il main crea una blackboard (inizialmente vuota), avvia il canale TV (specificando l'elenco di programmi da trasmettere), e poi avvia alcuni thread utenti (a distanza di tempo l'uno dall'altro):

```
import java.util.concurrent.ThreadLocalRandom;

public class BlackboardExample {
    public static void main(String[] args) throws InterruptedException {
        Blackboard<Programma> bbRai3 = new Blackboard<>();

        Programma[] progs = new Programma[] {
            new Programma("Meteo 3", "20:05", 5),
            new Programma("Blob", "20:10", 15),
            new Programma("TG3", "20:25", 40),
            new Programma("Dott. Stranamore", "21:05", 120),
            new Programma("Notturno", "23:05", 420)
        };
        new CanaleTv("Rai3", bbRai3, progs).start();

        for (int i = 0; i < 8; i++) {
            Thread.sleep(ThreadLocalRandom.current().nextInt(30));
            new Utente(bbRai3).start();
        }
    }
}
```

Un possibile output di questo programma è:

```
Utente Thread-1: attendo programma
Utente Thread-2: attendo programma
Rai3 trasmette Meteo 3 (inizia alle 20:05)
Utente Thread-1: sto guardando Meteo 3 (inizia alle 20:05)
Utente Thread-2: sto guardando Meteo 3 (inizia alle 20:05)
Rai3 trasmette Blob (inizia alle 20:10)
Utente Thread-3: attendo programma
Utente Thread-3: sto guardando Blob (inizia alle 20:10)
Rai3 trasmette TG3 (inizia alle 20:25)
Utente Thread-4: attendo programma
Utente Thread-4: sto guardando TG3 (inizia alle 20:25)
Utente Thread-5: attendo programma
Utente Thread-5: sto guardando TG3 (inizia alle 20:25)
Rai3 trasmette Dott. Stranamore (inizia alle 21:05)
Utente Thread-6: attendo programma
Utente Thread-6: sto guardando Dott. Stranamore (inizia alle 21:05)
Utente Thread-7: attendo programma
Utente Thread-7: sto guardando Dott. Stranamore (inizia alle 21:05)
Utente Thread-8: attendo programma
Utente Thread-8: sto guardando Dott. Stranamore (inizia alle 21:05)
Rai3 trasmette Notturno (inizia alle 23:05)
```

1.2 Varianti

Quali operazioni debbano essere bloccanti può dipendere dalla logica dell'applicazione. Ad esempio, si potrebbe:

- non voler sovrascrivere un messaggio che non è ancora stato letto da nessuno;
- non volersi bloccare sulla lettura;
- volersi bloccare in lettura solo per un certo tempo massimo;
- ecc.

2 Broadcast

Una comunicazione **broadcast** ("trasmissione"[1]) permette di inviare dati a più thread contemporaneamente. Solo i thread in attesa al momento dell'invio ricevono i dati.

[1] Un esempio classico di broadcast, che motiva la scelta del termine, è un trasmettitore radio (o TV), che comunica con un gran numero di ricevitori: tutti i ricevitori situati nella sua area di copertura ricevono il segnale, e il trasmettitore non può sapere con chi ha comunicato.

Sono possibili delle varianti, nelle quali si cambia (in base alle esigenze dell'applicazione) ciò che succede se, al momento dell'invio:

- non ci sono thread in attesa;
- un broadcast precedente è ancora in corso, cioè non tutti i thread in attesa hanno ricevuto i dati.

Nell'implementazione presentata in seguito, si sceglie di eseguire l'invio solo se ci sono thread in attesa e un eventuale messaggio precedente è stato ricevuto da tutti, altrimenti la chiamata `send` non fa niente (in particolare, non blocca il chiamante).

```
public class Broadcast<Data> {
    private Data message;
    // sending indica se c'è un messaggio non ancora ricevuto da tutti
    private boolean sending = false;
    private int waiting = 0;

    public synchronized void send(Data message) {
        if (waiting > 0 && !sending) {
            this.message = message;
            sending = true;
            notifyAll();
        }
    }

    public synchronized Data receive() throws InterruptedException {
        waiting++;
        try {
            while (!sending) {
                wait();
            }
        } finally {
            waiting--;
            if (waiting == 0) {
                // L'ultimo thread che riceve il messaggio
                // resetta il flag
                sending = false;
            }
        }
        return message;
    }
}
```

2.1 Esempio: agenzia di stampa

Un'agenzia di stampa, ANSA.it, fornisce le prime notizie su tutto quello che accade in Italia ai giornali che si abbonano al servizio. Per questo esempio, si suppone che ci siano 4 giornali interessati al servizio: IlCorriere.it, IlFatto.it, LaRepubblica.it e IlMattino.it.

L'invio di una notizia (messaggio) in broadcast è svolto da un apposito thread:

```
public class AnsaMessage extends Thread {
    private Broadcast<String> broadcast;
    private String message;

    public AnsaMessage(Broadcast<String> broadcast, String message) {
        this.broadcast = broadcast;
        this.message = message;
    }

    public void run() {
        System.out.println("News from ANSA.it: " + message);
        broadcast.send(message);
    }
}
```

Invece, i thread corrispondenti ai giornali aspettano di ricevere alcuni messaggi:

```
public class Newspaper extends Thread {
    private final String name;
    Broadcast<String> broadcast;

    public Newspaper(String name, Broadcast<String> broadcast) {
        this.name = name;
        this.broadcast = broadcast;
    }

    public void run() {
        for (int i = 0; i < 4; i++) {
            System.out.println(name + ": waiting for news...");
            try {
                String msg = broadcast.receive();
                System.out.println(name + ": " + msg);
            } catch (InterruptedException e) { break; }
        }
    }
}
```

Nel main:

1. si lancia un primo thread `AnsaMessage`, che cercherà di inviare un messaggio, ma, data l'assenza di thread in attesa di riceverlo, la chiamata `send` non avrà alcun effetto;
2. vengono avviati i thread corrispondenti ai giornali, che si mettono in attesa di ricevere messaggi;
3. si inviano altri messaggi.

```
public class BroadcastExample {
    public static void main(String[] args) throws InterruptedException {
        Broadcast<String> broadcast = new Broadcast<>();

        new AnsaMessage(
            broadcast, "Diminuiscono le immatricolazioni"
        ).start();
        Thread.sleep(100);

        new Newspaper("IlCorriere.it", broadcast).start();
        new Newspaper("IlFatto.it", broadcast).start();
        new Newspaper("LaRepubblica.it", broadcast).start();
        new Newspaper("IlMattino.it", broadcast).start();

        for (int i = 0; i < 5; i++) {
            Thread.sleep(10);
            new AnsaMessage(
                broadcast, "Aumentano le immatricolazioni"
            ).start();
            Thread.sleep(10);
            new AnsaMessage(
                broadcast, "Diminuiscono le immatricolazioni"
            ).start();
        }
    }
}
```

Un esempio di output è il seguente:

```
News from ANSA.it: Diminuiscono le immatricolazioni
IlCorriere.it: waiting for news...
IlFatto.it: waiting for news...
LaRepubblica.it: waiting for news...
IlMattino.it: waiting for news...
```

```
News from ANSA.it: Aumentano le immatricolazioni
News from ANSA.it: Diminuiscono le immatricolazioni
IlMattino.it: Aumentano le immatricolazioni
IlMattino.it: waiting for news...
IlFatto.it: Aumentano le immatricolazioni
IlFatto.it: waiting for news...
IlCorriere.it: Aumentano le immatricolazioni
IlCorriere.it: waiting for news...
LaRepubblica.it: Aumentano le immatricolazioni
LaRepubblica.it: waiting for news...
News from ANSA.it: Aumentano le immatricolazioni
IlMattino.it: Aumentano le immatricolazioni
IlCorriere.it: Aumentano le immatricolazioni
IlFatto.it: Aumentano le immatricolazioni
IlFatto.it: waiting for news...
IlMattino.it: waiting for news...
LaRepubblica.it: Aumentano le immatricolazioni
IlCorriere.it: waiting for news...
LaRepubblica.it: waiting for news...
...
```

2.2 Esempio: conferenze

Durante una conferenza, più speaker parlano in stanze diverse. (qui ne verrà simulata una sola).

- Gli ascoltatori entrano nella stanza dove si trattano gli argomenti di loro interesse, e aspettano lo speaker (se non è ancora arrivato).
- Lo speaker dà il messaggio di benvenuto a tutti i potenziali interessati, poi pronuncia un discorso.
- Il discorso consta di una serie di frasi, intervallate da pause.
- Gli ascoltatori possono alzarsi e andare via anche prima della fine del discorso.

La comunicazione tra lo speaker e gli ascoltatori avviene tramite un broadcast, creato nel main e passato a tutti i thread:

```java
public class Conferenza {
    public static void main(String[] args) throws InterruptedException {
        Broadcast<String> speechMessage = new Broadcast<>();

        System.out.println("La sala conferenze è aperta...");
        for (int i = 0; i < 10; i++) {
            new Ascoltatore(speechMessage).start();
```

```
        }

        Thread.sleep(1000);
        System.out.println("Entra lo speaker...");
        new Speaker(speechMessage).start();
    }
}
```

Ciascun thread `Ascoltatore` rimane ad ascoltare un numero casuale di frasi, poi esce:

```
import java.util.concurrent.ThreadLocalRandom;

public class Ascoltatore extends Thread {
    private Broadcast<String> speechMessage;

    public Ascoltatore(Broadcast<String> speechMessage) {
        setName("Ascoltatore " + getName());
        this.speechMessage = speechMessage;
    }

    public void run() {
        System.out.println(getName() + " in attesa di ascoltare");
        int numAscolti = ThreadLocalRandom.current().nextInt(10);
        for (int i = 0; i < numAscolti; i++) {
            String msg;
            try {
                msg = speechMessage.receive();
            } catch (InterruptedException e) { break; }
            System.out.println(getName() + " ha sentito: " + msg);
        }
        System.out.println(getName() + " esce");
    }
}
```

Il thread `Speaker` dà il benvenuto agli ascoltatori, e poi pronuncia il discorso:

```
public class Speaker extends Thread{
    private Broadcast<String> speechMessage;

    public Speaker(Broadcast<String> speechMessage) {
        setName("Speaker " + getName());
        this.speechMessage = speechMessage;
    }

    public void run() {
```

```
        System.out.println(getName() + " inizia a parlare...");
        speechMessage.send("Benvenuti!");
        for (int i = 0; i < 10; i++) {
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) { break; }
            speechMessage.send("Bla bla bla " + i);
        }
    }
}
```