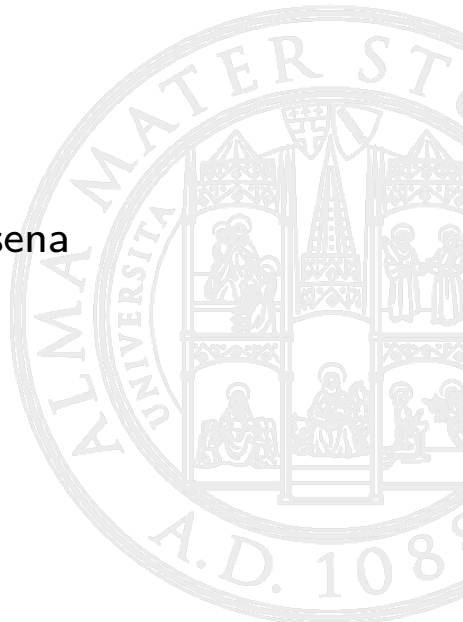
Tutorial AmbientTalk

F. Dicara G. Grossi M. Ricci
francesco.dicara@studio.unibo.it
gianluca.grossi2@studio.unibo.it
mattia.ricci8@studio.unibo.it

SISTEMI DISTRIBUITI
Laurea magistrale Ingegneria e Scienze informatiche
Alma Mater Studiorum - Università di Bologna sede di Cesena

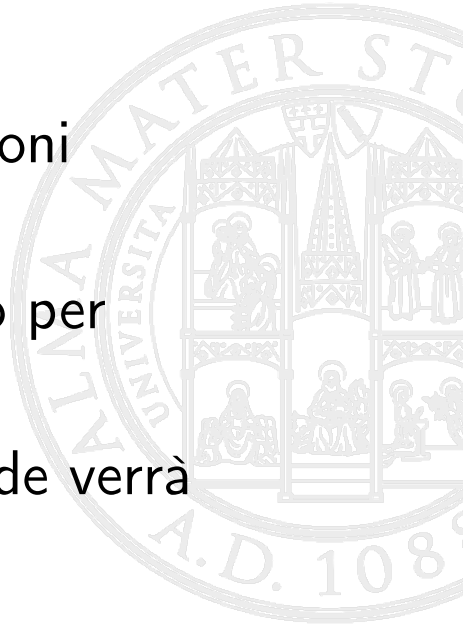
a.a. 2017/2018



Outline

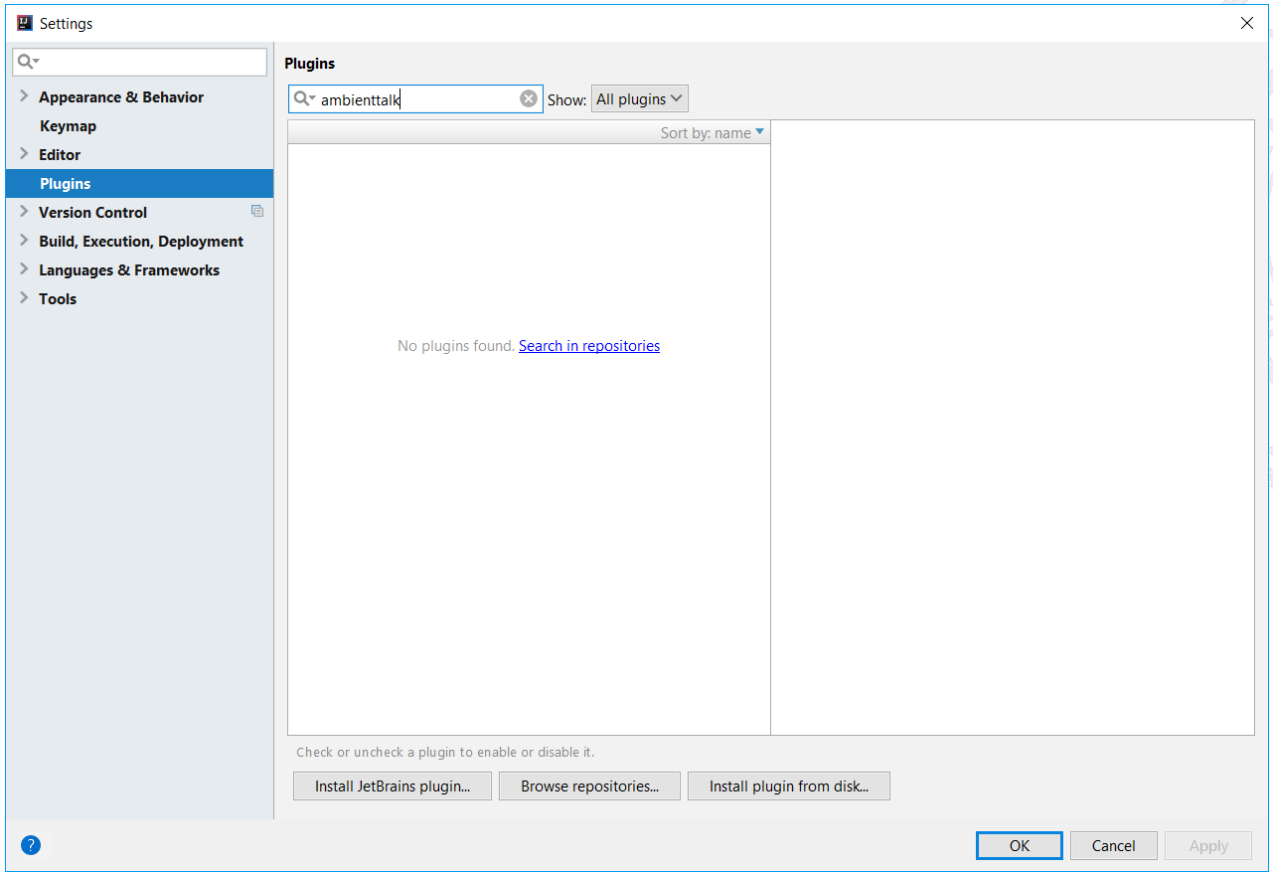
Plugin AmbientTalk per IntelliJ-IDEA

- In questa guida vedremo come sviluppare applicazioni distribuite nel linguaggio AmbientTalk.
- Per programmare utilizzeremo un plugin sviluppato per l'ambiente di sviluppo IntelliJ-IDEA.
- Dopo aver installato questo IDE, nelle prossime slide verrà mostrato come poter installare il plugin.



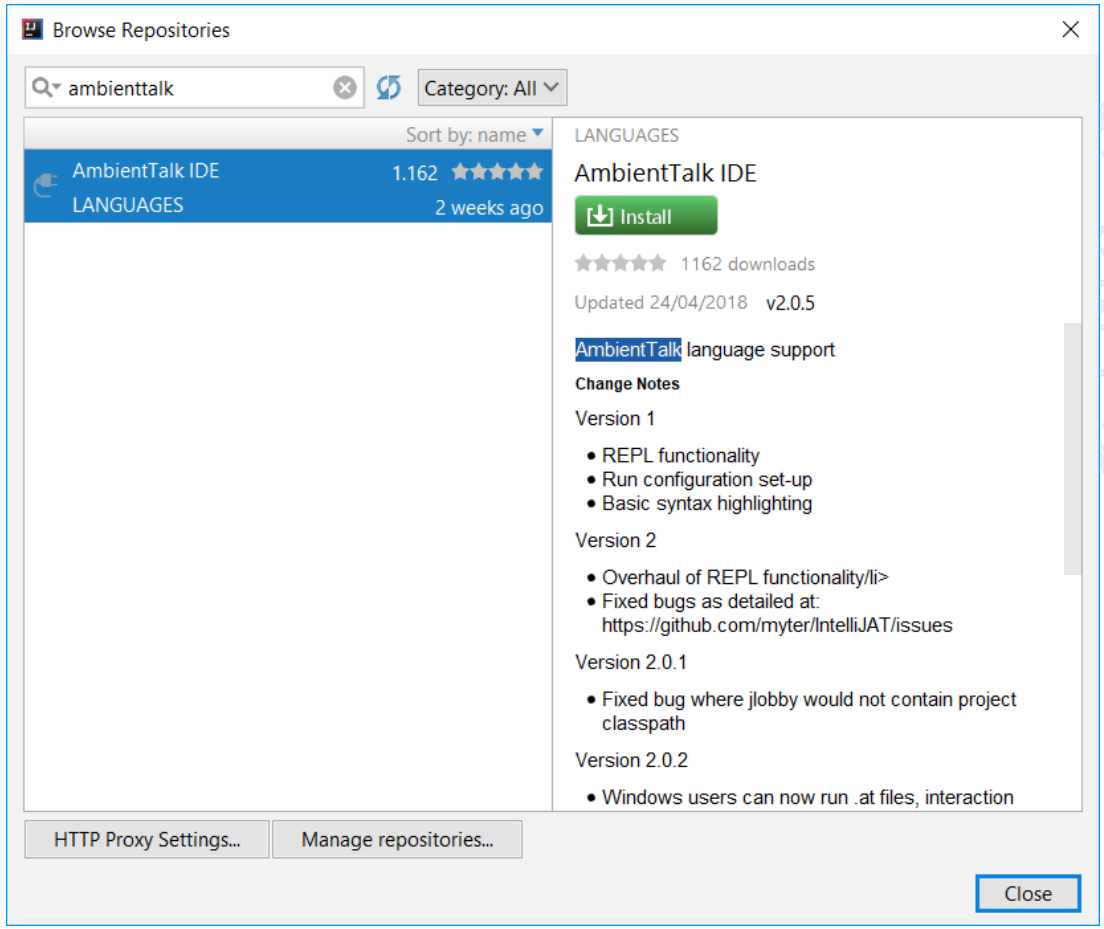
Installazione AmbientTalk (1)

1. Apriamo IntelliJ-IDEA, andiamo su File>Settings>Plugins e nell'area di ricerca digitiamo 'ambienttalk' e clicchiamo su 'Search in repositories':



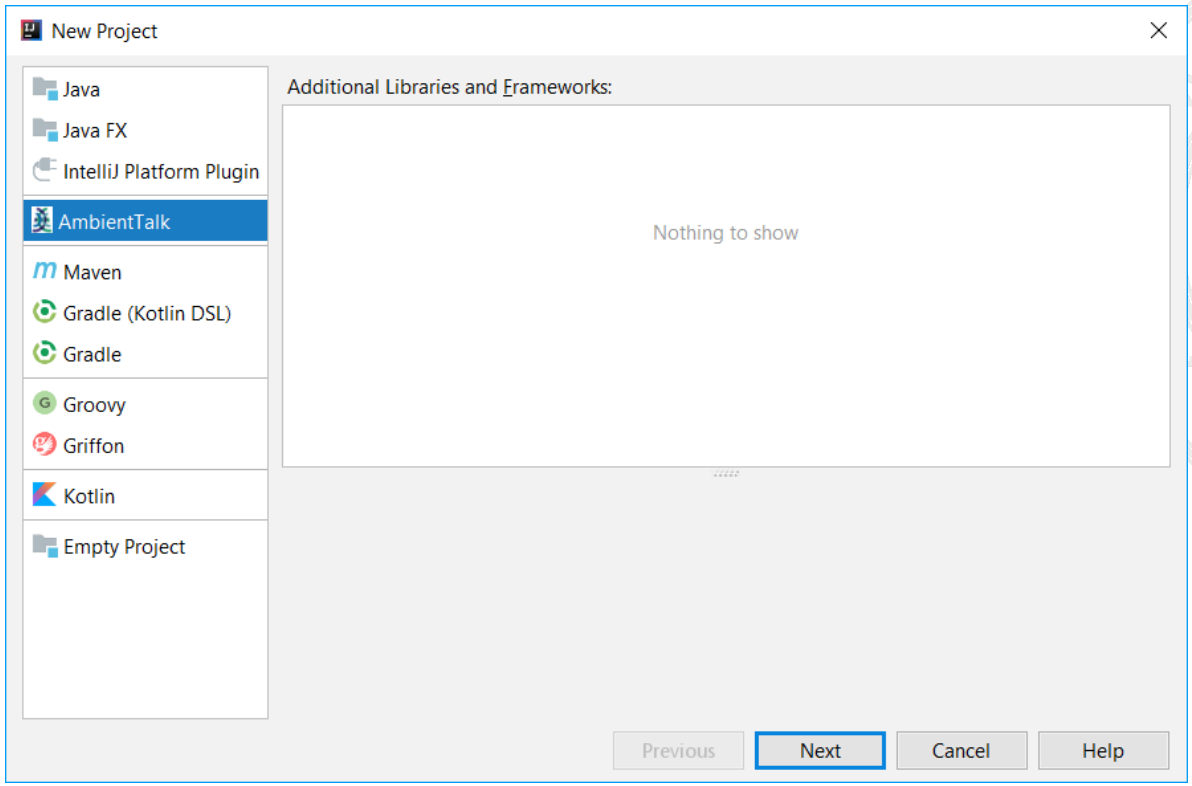
Installazione AmbientTalk (2)

2. A questo punto selezioniamo il plugin '*AmbientTalk IDE*' e clicchiamo sul pulsante verde '*Install*':



Installazione AmbientTalk (3)

3. Ora, riavviando IntelliJ, andiamo a creare un progetto di tipo AmbientTalk:



Prima esercitazione

- Lo scopo di questa prima esercitazione è di prendere confidenza con l'ambito sequenziale di AT: variabili, table, metodi, costrutti, oggetti...
- Per approfondire la sintassi del linguaggio, consultare il sito `http://soft.vub.ac.be/amop/at/tutorial/tutorial`
- Aprire in IntelliJ il materiale dedicato alla prima esercitazione...
- Nel sorgente sono già inseriti i test, con i quali controllare se ogni metodo realizzato è corretto.

Prima esercitazione: le basi (1)

- Metodo 1: Realizzare una funzione ricorsiva che restituisca in output il M.C.D. tra due numeri presi in input.

Prima esercitazione: le basi (1)

- Metodo 1: Realizzare una funzione ricorsiva che restituisca in output il M.C.D. tra due numeri presi in input.
- Suggerimento...
  - gcd(a,b) = a if a=b
  - gcd(a,b) = gcd(a-b,b) if a>b
  - gcd(a,b) = gcd(a,b-a) if b>a

Prima esercitazione: le basi (1)

- Metodo 1: Realizzare una funzione ricorsiva che restituisca in output il M.C.D. tra due numeri presi in input.
- Suggerimento...
  - gcd(a,b) = a if a=b
  - gcd(a,b) = gcd(a-b,b) if a>b
  - gcd(a,b) = gcd(a,b-a) if b>a
- Metodo 2: Scrivere una funzione che restituisca la lunghezza di una table (senza utilizzare il comando `table.length`).

Prima esercitazione: le basi (1)

- Metodo 1: Realizzare una funzione ricorsiva che restituisca in output il M.C.D. tra due numeri presi in input.
- Suggerimento...
  - gcd(a,b) = a if a=b
  - gcd(a,b) = gcd(a-b,b) if a>b
  - gcd(a,b) = gcd(a,b-a) if b>a
- Metodo 2: Scrivere una funzione che restituisca la lunghezza di una table (senza utilizzare il comando `table.length`).
- Metodo 3: Scrivere una funzione che restituisca il reverse di una table fornita in ingresso.

Prima esercitazione: le basi (2)

- Metodo 4: Scrivere la funzione `makeSetUnion` che prenda in ingresso due table e restituisca la concatenazione di queste, omettendo gli elementi duplicati.
  - Esempio: `makeSetUnion([1,2,4,4],[4,5])` -> output: `[1,2,4,5]`

Prima esercitazione: le basi (2)

- Metodo 4: Scrivere la funzione `makeSetUnion` che prenda in ingresso due table e restituisca la concatenazione di queste, omettendo gli elementi duplicati.
  - Esempio: `makeSetUnion([1,2,4,4],[4,5])` -> output: `[1,2,4,5]`
- Metodo 5: Scrivere la funzione `average`, che restituisca la media aritmetica dei numeri forniti in ingresso.

Prima esercitazione: le basi (2)

- Metodo 4: Scrivere la funzione `makeSetUnion` che prenda in ingresso due table e restituisca la concatenazione di queste, omettendo gli elementi duplicati.
  - Esempio: `makeSetUnion([1,2,4,4],[4,5])` -> output: `[1,2,4,5]`
- Metodo 5: Scrivere la funzione `average`, che restituisca la media aritmetica dei numeri forniti in ingresso.
- Metodo 6: Scrivere una funzione che prenda in ingresso una table e restituisca la table senza il primo elemento.

Prima esercitazione: object oriented (1)

Per testare i seguenti metodi occorre rimuovere `TODO` dal nome del test relativo

- Metodo 7: Implementare l'oggetto `Counter` con i metodi per incrementare, decrementare e ritornare il valore corrente del contatore.

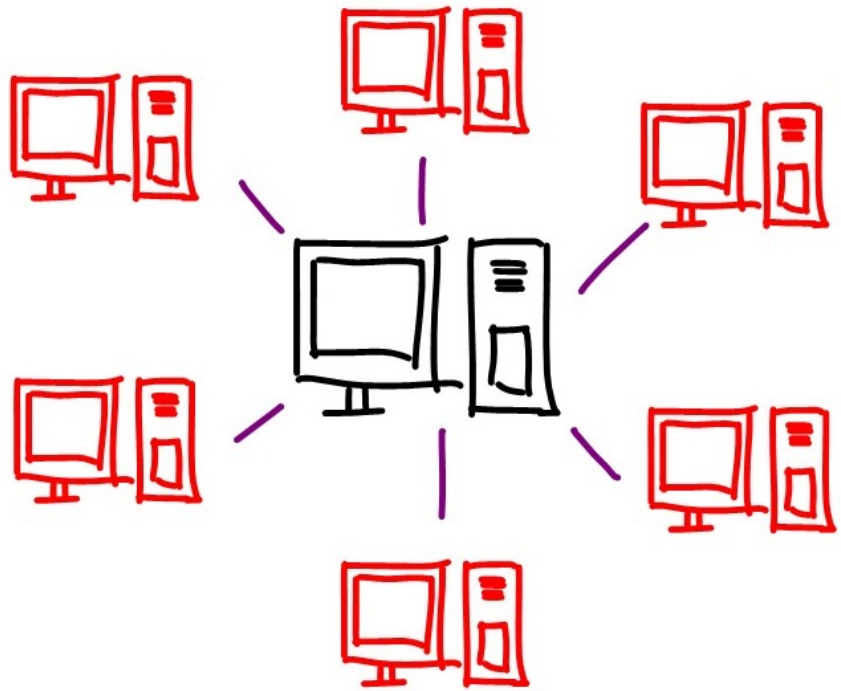
Prima esercitazione: object oriented (1)

Per testare i seguenti metodi occorre rimuovere `TODO` dal nome del test relativo

- Metodo 7: Implementare l'oggetto `Counter` con i metodi per incrementare, decrementare e ritornare il valore corrente del contatore.
- Metodo 8: Implementare l'oggetto `Polygon` con il metodo `perimeter`, che prende in ingresso un numero arbitrario di argomenti (corrispondenti alla misura dei lati) e restituisce il perimetro.
  - Estendere l'oggetto `Polygon` per creare l'oggetto `Rectangle`. Questo oggetto deve avere un costruttore, al quale viene passata l'altezza e la base del rettangolo, e due metodi per calcolare perimetro ed area.
  - Suggerimento 1: `def Object2 := extend: Object1 with: {...};`
  - Suggerimento 2: `super^metodoDaEstendere(...)`.

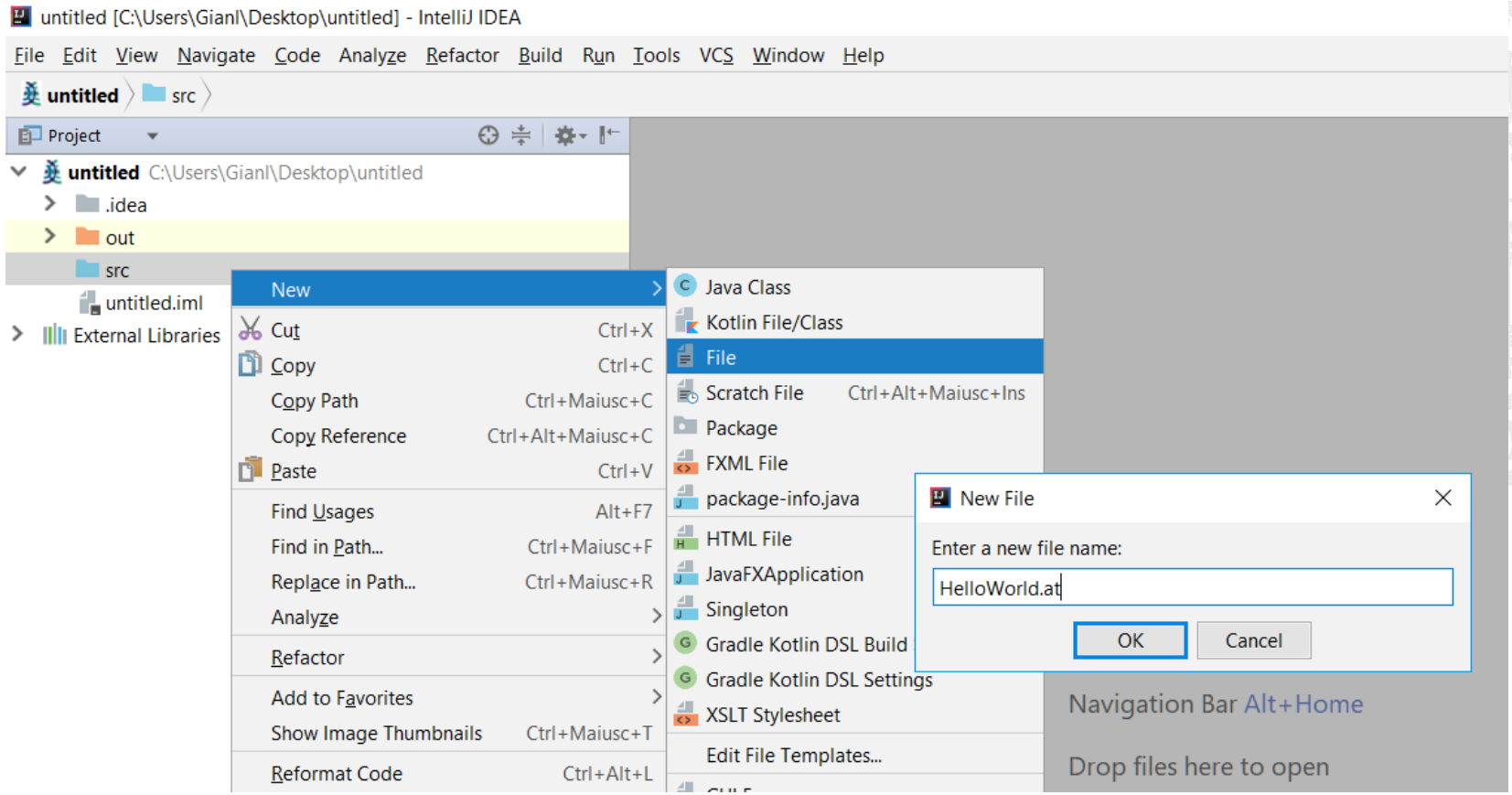
...verso il distribuito

Passiamo ora agli esercizi focalizzati sull'ambito distribuito!



Esercizio 1

- Creiamo un nuovo progetto AmbientTalk e, nella cartella *src*, andiamo a creare un file `HelloWorld.at`



Esercizio 1

- Lo scopo di questo esercizio è creare un'applicazione distribuita in cui più attori possono comunicare la loro presenza e trovare gli altri attori nella stessa rete.
- Nel file aperto precedentemente, definiamo il type tag che può essere scoperto nella rete usando la keyword `deftype`.
- Creiamo una funzione `makeGreeter(myName)` che prende in ingresso il nome dell'attore.
- All'interno della funzione creiamo un attore `actor:{...}`.
- Questo attore dovrà importare la libreria `/.at.lang.futures` che gli permetterà di utilizzare le future.

Esercizio 1

- L'attore dovrà:
  - Avere un metodo `getName()` che permetta di restituire il proprio nome;
  - Essere esportato nella rete usando il costrutto `export:as:`;
  - Gestire la scoperta di altri attori nella rete e agire di conseguenza (tramite il costrutto `whenever:discovered:`).
- Nella seguente slide verrà mostrato il codice ad esclusione della logica dell'attore.

Soluzione incompleta Es.1

```
/* Definisce il tipo che puo' essere scoperto nella rete */
deftype Greeter;

def makeGreeter(myName) {
    // Crea un'attore
    actor: {
        // Gli attori hanno un namespace separato, importiamo le future
        import /.at.lang.futures;

        // Un metodo che puo' essere chiamato dagli altri attori
        def getName(){myName};

        // Esportiamo questo attore sulla rete
        export: self as: Greeter;

        // Logica Principale: se viene scoperto un altro attore allora creiamo
        //una future che aspetta il nome di un altro attore nella rete...
        whenever: Greeter discovered: {|other|
            //TODO
        };
    };
};

// Creiamo 2 attori che si scambieranno il saluto
makeGreeter("Alice");
makeGreeter("Bob");
```

Soluzione completa Es.1

```
deftype Greeter;

def makeGreeter(myName) {
    actor: {
        import /.at.lang.futures;

        def getName(){myName};

        export: self as: Greeter;

        whenever: Greeter discovered: {|other|
            // Prendiamo il nome dell'attore trovato nella rete in maniera
            // asincrona e stampiamo un messaggio
            when: other<-getName()@FutureMessage becomes: {|name|
                system.println("Hello" + name + "from" + myName);
            };
        };
    };
};

makeGreeter("Alice");
makeGreeter("Bob");
```

Esercizio 2

- Immaginiamo di avere una tavola rotonda con `n` partecipanti;
- Ogni partecipante deve scambiare un messaggio contente un intero con il partecipante successivo;
- Ogni volta che il partecipante riceve un messaggio, incrementa il valore di 1.

Esercizio 2

- Immaginiamo di avere una tavola rotonda con `n` partecipanti;
- Ogni partecipante deve scambiare un messaggio contente un intero con il partecipante successivo;
- Ogni volta che il partecipante riceve un messaggio, incrementa il valore di `1`.

Nota bene:

- I messaggi devono essere scambiati tra i partecipanti sempre nello stesso ordine (ad anello dal numero `0` al numero `n-1`);
- I messaggi dovranno essere scambiati tra gli attori solo dopo aver verificato che tutti gli `n` attori sono sulla rete.

Soluzione incompleta Es.2 (1)

```
1   def makeTeamMember(id,ringSize) {
2           actor: { |id,ringSize|
3                   import /.at.lang.futures;
4                   deftype Member;
5                   def [fut,res] :=  makeFuture();
6
7                   def interface := object: {
8                           def getId() { id };
9
10                          def next(count) {
11                                  system.println("Id=" + id + ",count=" + count);
12                                  fut<-next( count + 1 );
13                          };
14                  };
15
16                  def next() {
17                          interface<-next(0);
18                  };
19
20                  export: interface as: Member;
21                  whenever: Member discovered: { |mem|
22                          /TODO
23                  };
24          };
25  };
26
27  def ringSize := 10;
28  def c := -1;
29  def members[ringSize] { c := c+1; makeTeamMember(c,ringSize);  };
30  members[1]<-next();
```

Soluzione incompleta Es.2 (1)

- Nel corpo degli attori viene gestita la comunicazione tra i peer;
- Nella riga 29 viene creata una `table` che rappresenta tutti i peer che dovranno comunicare;
- Nella riga successiva viene avviato il primo peer;
- Implementare il costrutto `whenever:discovered:` per consentire lo scambio di messaggi solo dopo aver verificato che tutti i peer sono in rete.

Soluzione Es.2

```
whenever: Member discovered: { |mem|
  when: mem<-getId()@FutureMessage becomes: { |rid|
    if: (rid == ((id+1)%ringSize)) then: {
          res.resolve(mem);
    }
  };
};
```

Esercizio 3

- Lo scopo di questo esercizio è entrare all'interno del comportamento delle far references e della gestione dei guasti implementando un player musicale da usare sui dispositivi mobili.
- Il player musicale contiene una libreria di canzoni (sotto forma di stringa); quando viene scoperto un altro peer nel suo ambiente loro si scambiano la lista della canzoni.
- Dopo lo scambio dovrà essere calcolata la percentuale di canzoni che gli utenti hanno in comune.

Testo Es.3

```
def makeMusicPlayer(userName) {
    actor: {
        import /.at.lang.futures;

        def Vector := jlobby.java.util.Vector;
        def myLib := Vector.new();  //libreria musicale del peer

        def interface := object: {
            // ritorna un oggetto sessione incapsulando lo stato del processo
            //di scambio della libreria musicale
            def openSession(remoteUser) {
                /*...*/
            };
        };

        // invocando questo metodo il player inizia a scoprire altri peer
        def goOnline() {
            /*...*/
        };

        def addSong(song) {
            myLib.add(song);
        };
    };
};
```

Esercizio 3

- Un nuovo player musicale può essere creato chiamando il metodo `makeMusicPlayer(userName)`.
- Questa funzione definisce un oggetto innestato chiamato `interface` che implementa il protocollo distribuito che funziona in questo modo:
  1. Una volta che un player musicale ne scopre un altro, chiede al player remoto di aprire una sessione di scambio canzoni mandando un messaggio `openSession` che restituisce un oggetto sessione.
  2. Un oggetto sessione implementa i metodi per mandare le canzoni (`uploadSong()`) e per segnalare che tutte le canzoni sono state inviate (`endExchange()`).
  3. Dopo lo scambio, viene calcolata la percentuale di canzoni che i player musicali hanno in comune.

Implementazione interface Es.3

```
def interface := object: {
  def openSession(remoteUser) {
    system.println("APRO LA SESSIONE");
    def senderLib := Vector.new(); // memorizza la libreria del mittente
    def session := object: {
      def uploadSong(song) {
        senderLib.add(song);
        "ok"; // ack per confermare la ricezione
      };
      def endExchange() {
       system.println("MITTENTE:'"+remoteUser+"',DESTINATARIO:'"+userName+"'");
       senderLib.retainAll(myLib); //elimina le canzoni in comune
       def matchRatio := (senderLib.size()*100/(myLib.size()+0.01)).round();
       system.println("COMPATIBILITA':"+ matchRatio+"%");
      };
    };
  };
};
```

Implementazione goOnline() Es.3

```
def goOnline() {
 deftype MusicPlayer;
 export: interface as: MusicPlayer;
 whenever: MusicPlayer discovered:{ |musicPlayerReference|
   system.println("SCOPERTO UN MUSIC PLAYER: " + musicPlayerReference);
   //quando scopro un nuovo music player creo un nuovo oggetto sessione
   def fut := musicPlayerReference<-openSession(userName)@FutureMessage;
   when: fut becomes: {  |session|
     //invio canzoni chiamando il metodo uploadSong() del peer remoto
     def iterator := myLib.iterator();
     def sendSongs() {
       if: (iterator.hasNext()) then: {
        def song := iterator.next();
        when: session<-uploadSong(song)@FutureMessage becomes: { |ack|
          sendSongs(); // chiamata ricorsiva
        } catch: { |e| system.println("ERRORE:" + e) };
       } else: {
        //segnalazione della fine dello scambio
        session<-endExchange();
       };
       //nelle chiamate ricorsive (sendSong()) meglio ritornare nil
       nil;
      };
      system.println("INIZIO A INVIARE CANZONI");
      sendSongs();
     } catch: TimeoutException using: { |e|
      system.println("NON POSSO APRIRE UNA SESSIONE");
   };
 };
};
```

Gestione disconnessioni Es.3

- All'interno del metodo `goOnline()` e del costrutto `whenever:discovered:` possiamo aggiungere i due costrutti `whenever:disconnected:` e `whenever:reconnected:` per gestire le disconnessioni e riconnessioni dei peer dalla rete.

```
whenever: musicPlayerReference disconnected: {
    system.println("MUSIC␣PLAYER␣"
      + musicPlayerReference + "␣DISCONNESSO");
};
whenever: musicPlayerReference reconnected: {
    system.println("MUSIC␣PLAYER␣"
      + musicPlayerReference + "␣RICONNESSO");
};
```

Avviamo il programma Es.3

- Ora che il nostro codice è completo possiamo creare i peer.
  Esempio: `peerA.makeMusicPlayer("A");`
  `peerB.makeMusicPlayer("B");`
- Successivamente possiamo aggiungere canzoni alla libreria.
  Esempio: `peerA.addSong("nome-canzone");`
  `peerB.addSong("nome-canzone");`
- Infine rendiamo online sulla rete il peer appena creato.
  Esempio: `peerA.goOnline();`
  `peerB.goOnline();`.

Soluzioni complete esercizi distribuiti

Le soluzioni complete dei 3 esercizi distribuiti sono disponibili nel file zip.

Creare un progetto AmbientTalk su Android (1)

- Un'applicazione AmbientTalk consiste in un progetto Android contenente il codice della GUI Android e lo script AmbientTalk come risorsa.
- Inoltre, il progetto Android deve includere la libreria *android-core*, ovvero un wrapper in grado di impostare l'interprete di AmbientTalk ed eseguirne gli script.
- Passo passo spieghiamo tutte le fasi da svolgere:
  - Creiamo un nuovo progetto Android;
  - Importiamo il progetto *android-core* che si trova all'indirizzo `https://github.com/ferranDelgado/weScribble/tree/master/android-core` (ricordiamo di importare anche tutti i file `.jar` del progetto *android-core*);
  - Creiamo una cartella `atlib` sotto la cartella `assets`, dove salveremo tutti i nostri file AmbientTalk. (Ad esempio, un file AT caricato come `\.demo.weScribble` dovrebbe essere memorizzato sotto `assets/atlib/demo/weScribble.at`).

Creare un progetto AmbientTalk su Android (2)

- Affinché la nostra applicazione possa accedere alla libreria di AmbientTalk, dovremo creare una sottoclasse dell'activity `AssetInstaller` (fornita dalla libreria di *android-core*) e avviarla in una delle activity Android;
- L'activity creata dovrà, ovviamente, essere registrata nel `manifest` del progetto;
- Nel `manifest` è inoltre necessario aggiungere i seguenti permessi:

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Creare un progetto AmbientTalk su Android (3)

- Una volta implementata, per avviare la nostra applicazione dovremo usare l'*IAT* (AmbientTalk Interactive Console).
- E' consigliato estendere la classe `AsyntTask` per lanciare l'*IAT*, per esempio:

```
public class StartIATTask extends AsyncTask<Void, Void, Void> {
    @Override
    protected Void doInBackground(Void... arg0) {
        try {
            iat_ = IATAndroid.create(WeScribbleActivity.this);
            ATWeScribble atws = (ATWeScribble)
                iat_.evalAndWrap("/.demo.weScribble.weScribble.return",
                    ATWeScribble.class);
            atws.handshake(WeScribbleActivity.this);
        } catch (Exception e) {
            Log.d("portal-pong", "Could not create IAT", e);
        }
        return null;
    }
}
```

Creare un progetto AmbientTalk su Android (4)

- Nel frammento di codice della slide precedente, `ATWeScribble` è un'interfaccia Java racchiusa in un oggetto AmbientTalk chiamato `return`, che viene definito come segue:

```
def return() {
    def localInterface := object: {
        def handshake(act) {
            androidActivity := act;
        };
        /*...*/
    };
};
```

- `return` deve restituire l'interfaccia locale dei metodi AmbientTalk che possono essere chiamati dal codice Android (come dall'esempio precedente).
- Utilizzando il metodo `handshake`, il codice AmbientTalk può ottenere un riferimento all'activity Android.

WeScribble: AmbientTalk Android application

WeScribble è un'applicazione AmbientTalk per Android che rende possibile partecipare a un disegno in collaborazione con altre persone su dispositivi differenti.
Quando più persone che utilizzano *weScribble* entrano nel raggio di comunicazione, possono contemporaneamente disegnare su un foglio di disegno virtualmente condiviso tra i diversi partecipanti.