A Brief Introduction to Software Agents and Their Programming Languages

Giuseppe Petrosino, Università degli Studi di Modena e Reggio Emilia

Five trends in computing

- Ubiquity - computer systems everywhere
- Interconnection - distributed systems is the norm
- Intelligence - more complex automated tasks
- Delegation - more control given to computer systems
- Human-Orientation - paradigms for interactions with machines are closer to humans

We need ...

- Ubiquity
- Interconnection

… *systems able to exploit ubiquitous processing power*

- Intelligence
- Delegation
- Human-Orientation

… *systems able to act on our behalf*

- able to operate *independently*
- able to act to *represent our best interests*

What is an Agent?

Anything that

- perceives its environment (*sensors*)
- acts on the environment (*actuators*)
- is autonomous (has control over its state and execution)
- may exhibit (some degree of)
    - social abilities - to communicate with other agents
    - reactivity - ability to react to changes in the environment
    - proactivity - ability to take initiative, acting in advance to a future situation
    - rationality - to figure out what to do, understand the environment
    - learning abilities - to improve its performance
    - other components of intelligence - understand images, natural language etc…

What is a Multi-Agent System?

- A system of several agents
- Agents interact with one another by exchanging messages
- They operate to achieve their own interests and goals
    - Which could differ!

$\searrow$ *coordinate, cooperate, compete, negotiate*

Autonomous Agents and Multi-agent Systems

- **Agent** design
    - How do we build agents capable of independent, autonomous action, in order to successfully carry out tasks we delegate to them?
- **Society** design
    - How do we build systems made of agents, that are capable of interacting, and which may not share the same interests/goals, in order to successfully carry out tasks we delegate to them?

(not orthogonal problems!)

Autonomous Agents and Multi-agent Systems

- How can interacting agents can understand each other?
  - What sorts of common languages can agents use to communicate?
  - How can agents reason about the consequences of their interactions?
  - How can we make sure that when they communicate wrt some concept, they share the same interpretation of this concept?

AAMAS and other fields

- **Artificial Intelligence**
- **Distributed Systems**
- **Economics**
- **Philosophy**
- **Logic**
- **Ecology**
- **Game Theory**
- **Social Sciences**

- Both a strength…
    - the field can take inspiration from theory and methodologies of the other fields
- … and a weakness
    - many perspectives and views on the core aspects of the field

Agents as a paradigm for software engineering

- We could use agents and MAS to model computation as a **process of interaction**
  - We can understand systems composed of passive objects with a state, on which we perform operations
  - We can understand systems made of pure functions that produce an output when evaluated on an input
  - … we can also understand systems made of autonomous agents interacting with each other

IEEE FIPA

Foundation for Intelligent Physical Agents

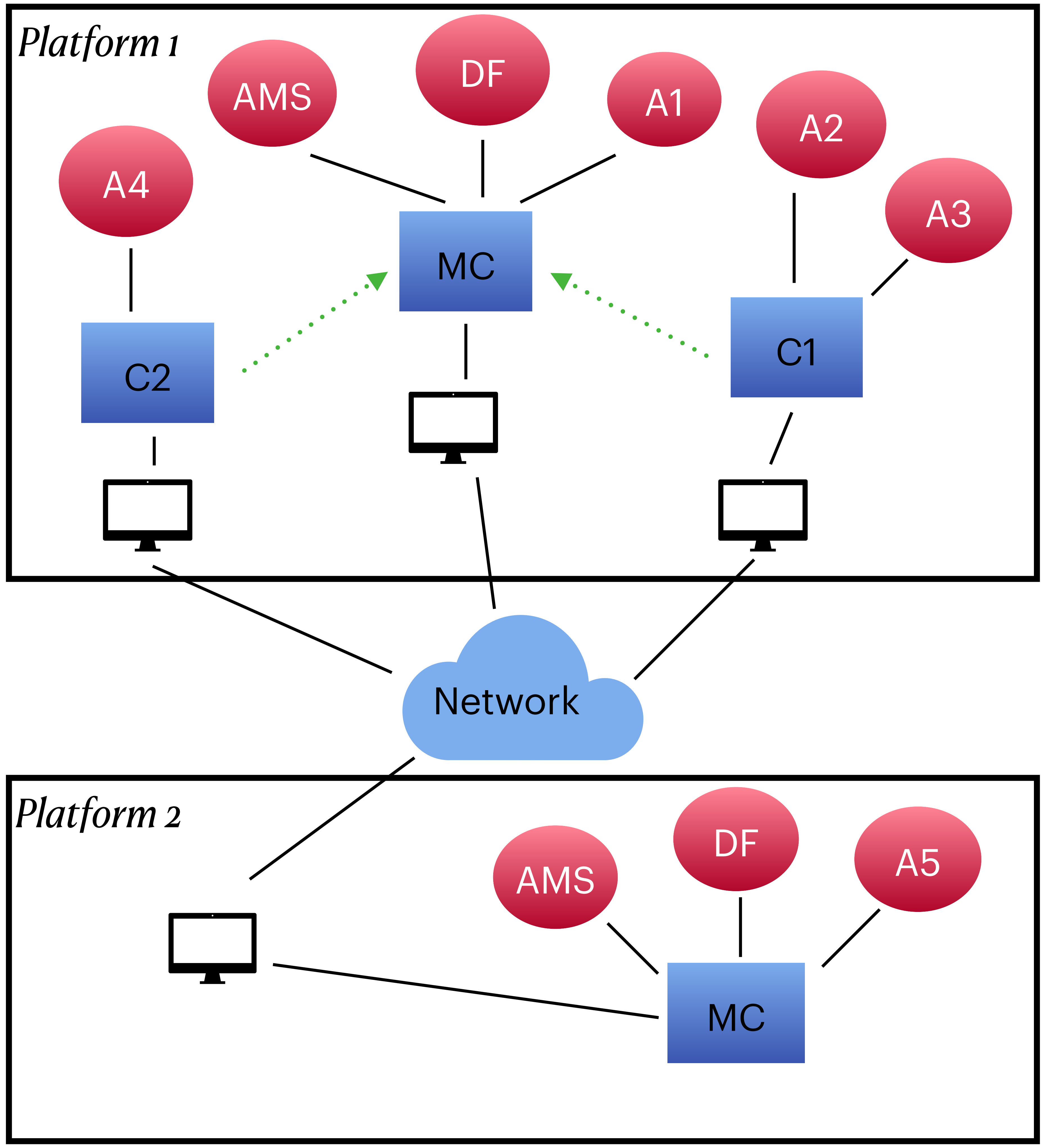
- International non-profit association
  - more than 60 members from 20 countries
  - companies, research centres, universities
- Contributed to promote research and innovation on software for about a decade (1996-2006)
- Goal: establishing a set of interoperability specifications for heterogeneous agents
  - message transport (transport protocols, envelope / content representations)
  - agent communication (content languages, communicative acts, interaction protocols)
  - agent management, abstract architecture, application domains

Java Agent DEvelopment Framework - JADE

- development started in 1998, by Telecom Italia
- it complies FIPA specifications
    - it became its *de facto* reference implementation
- JADE provided the practical means to use agents as software components
- It helped identify several agent-oriented abstractions
    - e.g., platforms, containers, agents, behaviours, ontologies

JADE: (agent) platforms and containers

- JADE Agents are software entities running in a runtime environment made of
    - (agent) platforms
    - (agent) containers
- A platform is made of
    - one main container
    - several peripheral containers
- Containers provide essential services to agents
    - e.g., message transport



JADE: Agents

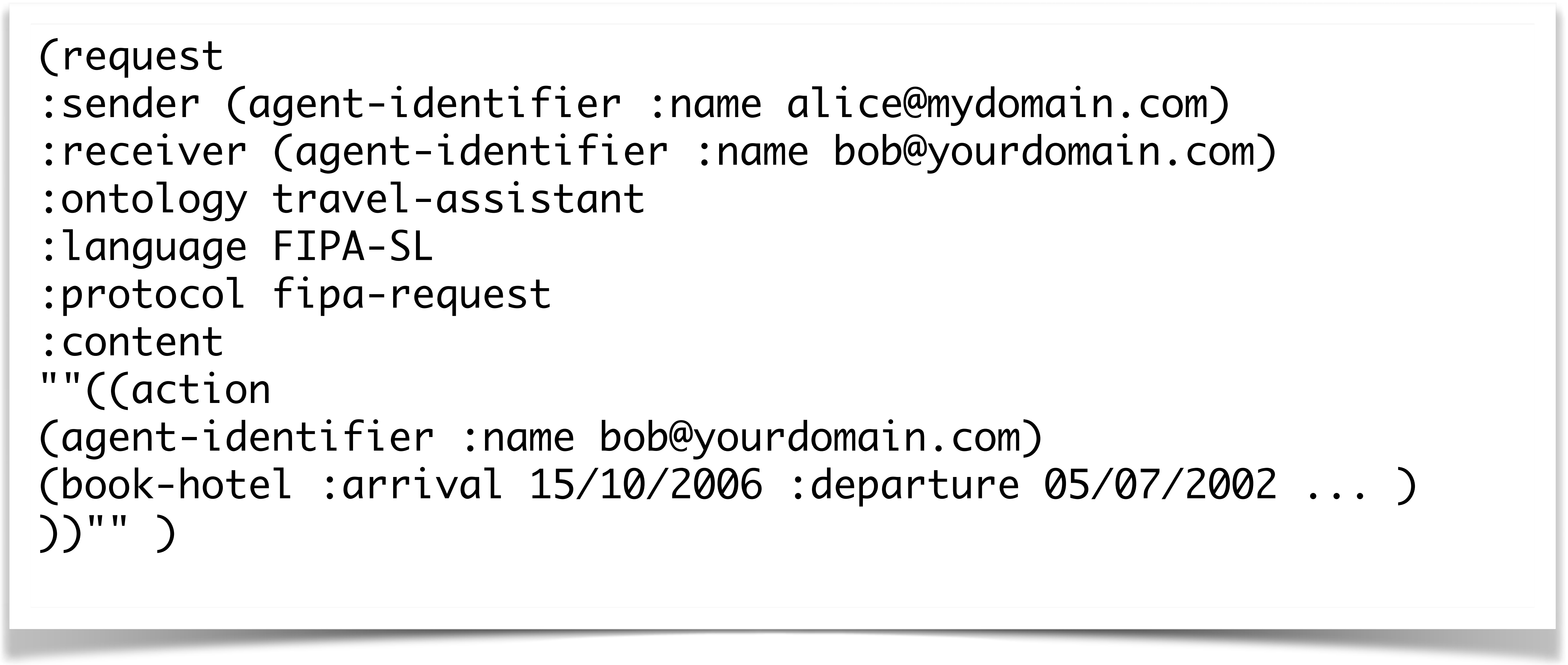
- single-threaded
- identified by unique (within the same platform) strings of characters (Agent IDentifiers, AID)
- interact by exchanging FIPA ACL messages
    - they refer to each other with AIDs
- has a lifecycle made of several states, including
    - initialized
    - active
    - waiting
    - deleted

JADE: Behaviours

- Abstraction to manage procedures agents follow to perform tasks
- Each agent engages one or more behaviours during its execution
    - concurrent execution, cooperative (non--preemptive) scheduler
- Main entry point: `void action()` method; termination condition: `boolean hasDone()` method
- Various types of predefined behaviours
    - Simple behaviours: `CyclicBehaviour`, `OneshotBehaviour`
    - Composite behaviours: `SequentialBehaviour`, `ParallelBehaviour`, …
    - Custom behaviours: just extend `jade.core.behaviours.Behaviour`!

JADE: Messages

- delivery of message is transparent to agents
    - programmers do not need to worry about connection, addresses, hosts
- messages (FIPA-ACL Message Structure Specification - SC00061) are associated with:
    - a FIPA **performative**
    - a sender AID
    - a receiver AID
    - a FIPA ACL language
    - an **ontology**
    - a message content
    - others (protocol, conversation identifier, ...)



```
(request
:sender (agent-identifier :name alice@mydomain.com)
:receiver (agent-identifier :name bob@yourdomain.com)
:ontology travel-assistant
:language FIPA-SL
:protocol fipa-request
:content
""((action
(agent-identifier :name bob@yourdomain.com)
(book-hotel :arrival 15/10/2006 :departure 05/07/2002 ... )
))"" )
```

JADE: Performative

- FIPA defines communication in terms of communicative acts (CA)
    - based on *Speech Act Theory* (Searle, 1969)
- Each message can perform several functions
    - e.g., B sends to A that he *agrees* to perform the requested action H
        - phatic (used to maintain the communication) function of agreeing to proceed
        - paralinguistic (used to relate a message to another message) function of referring to another CA
- Performatives are used in JADE to specify the kind of functions CAs intend to perform
    - Standardized by FIPA Communicative Act Library Specification (SC00037)

JADE: Ontologies

- For FIPA, just tags attached to messages
- For JADE, they are more
  - Ontologies are an abstractions that help programmers to define, create and serialize/deserialize the contents of messages
  - Ontologies are composed of
    - concepts - entities made of properties
    - (agent) actions - to refer to actions that could be performed by agents
    - predicates/propositions - to create logical expressions
- Agents can register/deregister ontologies dynamically

Agent-Oriented Programming

- "*a new programming paradigm, based on a societal view of computation*" (Yoav Shoam)
  - When Shoam introduced the term, it also introduced his own AOP language (AGENT-0)
- AOP is equally related to
  - the abstractions used to construct agents
  - to the syntax and semantics adopted to manage the abstractions

    ➡ Agent-Oriented Programming languages

Jadescript

- Agent development with JADE & Java is often perceived as difficult for newcomers
- Jadescript: programming language to develop JADE agents
    - Compiles to Java sources
    - Static typing (with simple type inference)
    - Syntax inspired from agent pseudocode
    - Created with Xtext (compiler, Eclipse IDE plugin, syntax-highlighting, autocompletion…)
    - Main abstractions inspired from / based on JADE abstractions
- Under development

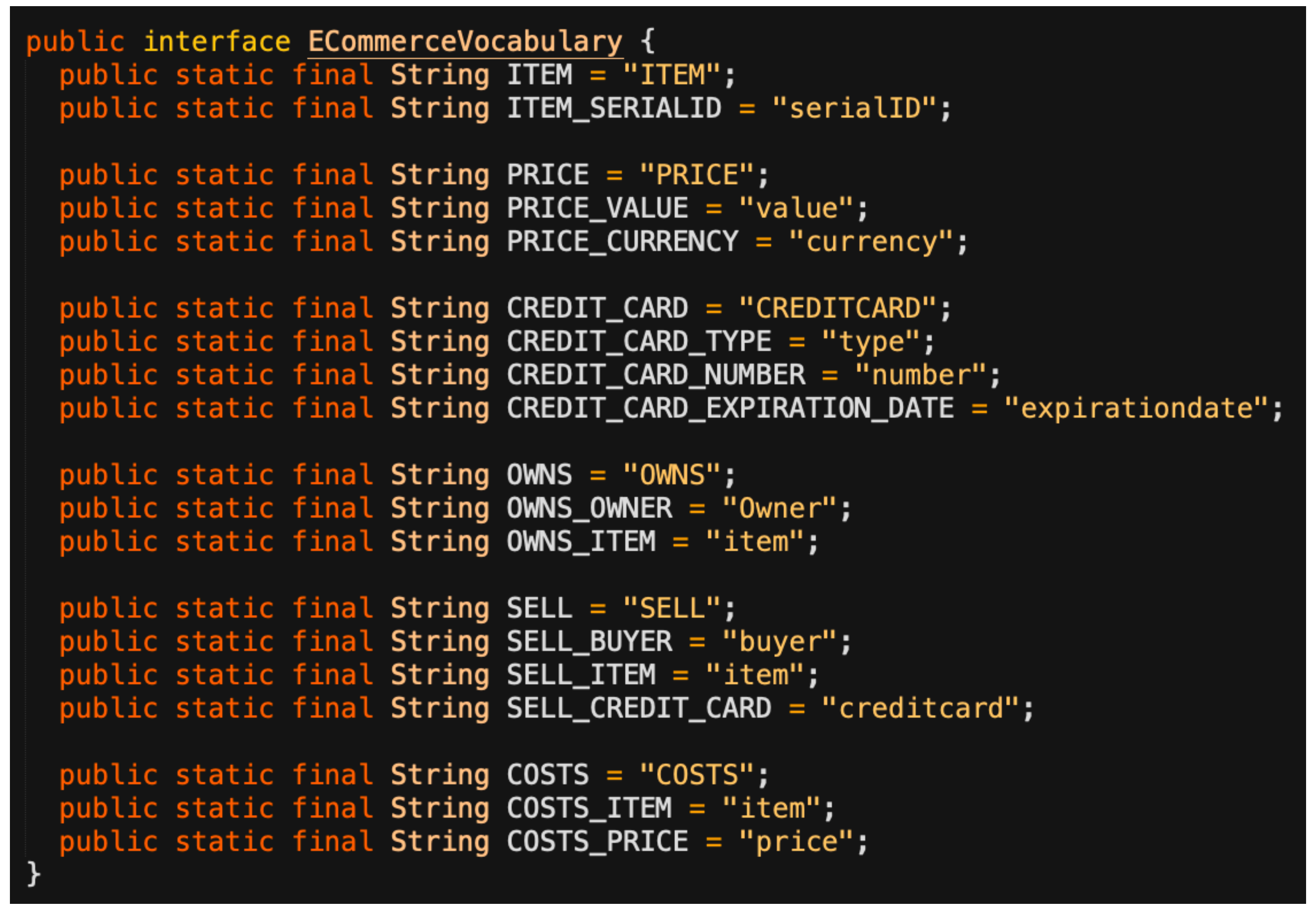
Jadescript: Advantages

- Higher level of abstraction, constructs related to AOP are first-class citizens
- The compiler...
  - ... checks the code to prevent bugs and guide development
  - ... enforces and internalize good JADE development practices
    - e.g., putting an agent to waiting state, using and registering ontologies
  - ... reduces boilerplate
    - in e.g., defining ontologies, sending and receiving messages
- Much higher readability (than Java+JADE)!

Example: E-commerce Ontology

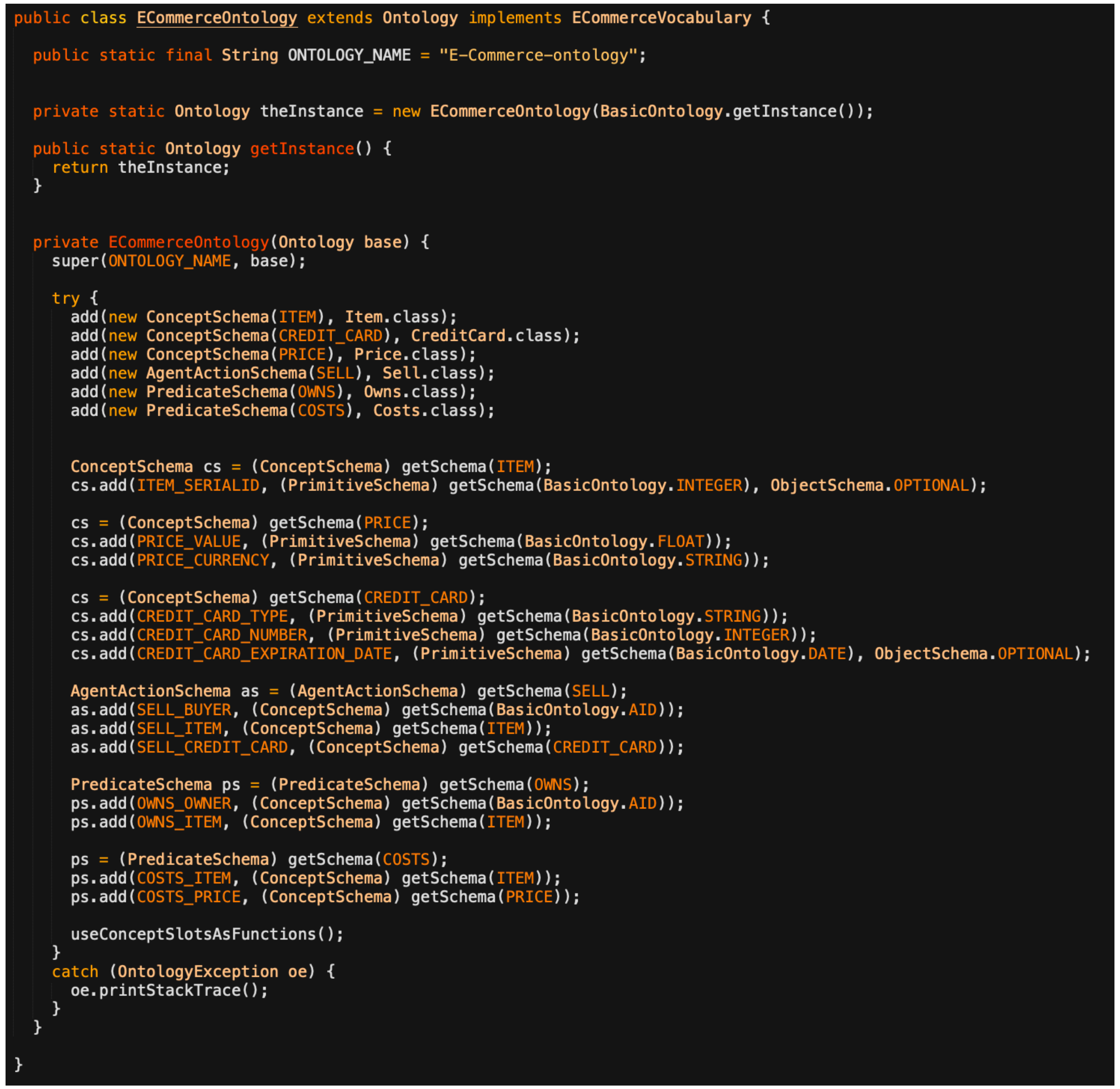
- 3 concepts
  - Item (serial ID)
  - CreditCard (type, number, expiration date)
  - Price (value, currency)
- 1 agent action
  - Sell (buyer, item, credit card)
- 2 predicates
  - Owns (owner, item)
  - Costs (item, price)

Example: JADE E-commerce Ontology (1)



```java
public interface ECommerceVocabulary {
  public static final String ITEM = "ITEM";
  public static final String ITEM_SERIALID = "serialID";

  public static final String PRICE = "PRICE";
  public static final String PRICE_VALUE = "value";
  public static final String PRICE_CURRENCY = "currency";

  public static final String CREDIT_CARD = "CREDITCARD";
  public static final String CREDIT_CARD_TYPE = "type";
  public static final String CREDIT_CARD_NUMBER = "number";
  public static final String CREDIT_CARD_EXPIRATION_DATE = "expirationdate";

  public static final String OWNS = "OWNS";
  public static final String OWNS_OWNER = "Owner";
  public static final String OWNS_ITEM = "item";

  public static final String SELL = "SELL";
  public static final String SELL_BUYER = "buyer";
  public static final String SELL_ITEM = "item";
  public static final String SELL_CREDIT_CARD = "creditcard";

  public static final String COSTS = "COSTS";
  public static final String COSTS_ITEM = "item";
  public static final String COSTS_PRICE = "price";
}
```

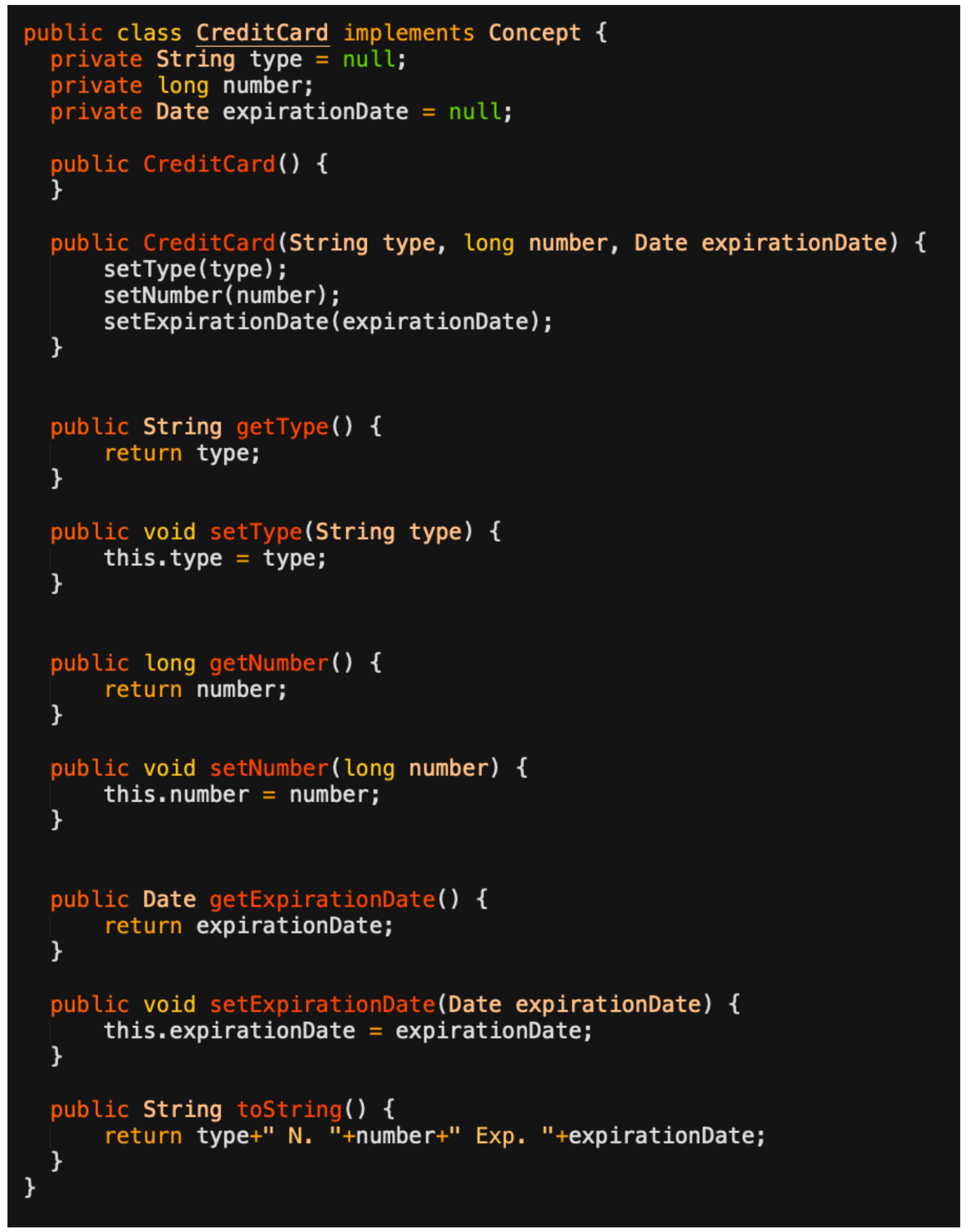
Vocabulary = 26 LOC



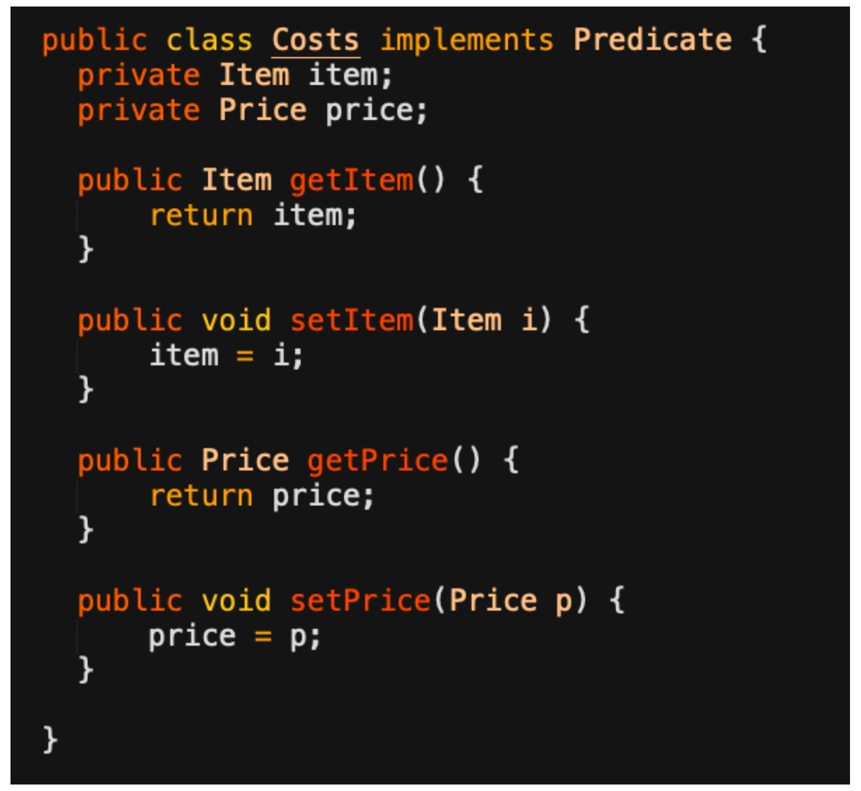
```java
public class ECommerceOntology extends Ontology implements ECommerceVocabulary {

  public static final String ONTOLOGY_NAME = "E-Commerce-ontology";


  private static Ontology theInstance = new ECommerceOntology(BasicOntology.getInstance());

  public static Ontology getInstance() {
    return theInstance;
  }


  private ECommerceOntology(Ontology base) {
    super(ONTOLOGY_NAME, base);

    try {
      add(new ConceptSchema(ITEM), Item.class);
      add(new ConceptSchema(CREDIT_CARD), CreditCard.class);
      add(new ConceptSchema(PRICE), Price.class);
      add(new AgentActionSchema(SELL), Sell.class);
      add(new PredicateSchema(OWNS), Owns.class);
      add(new PredicateSchema(COSTS), Costs.class);


      ConceptSchema cs = (ConceptSchema) getSchema(ITEM);
      cs.add(ITEM_SERIALID, (PrimitiveSchema) getSchema(BasicOntology.INTEGER), ObjectSchema.OPTIONAL);

      cs = (ConceptSchema) getSchema(PRICE);
      cs.add(PRICE_VALUE, (PrimitiveSchema) getSchema(BasicOntology.FLOAT));
      cs.add(PRICE_CURRENCY, (PrimitiveSchema) getSchema(BasicOntology.STRING));

      cs = (ConceptSchema) getSchema(CREDIT_CARD);
      cs.add(CREDIT_CARD_TYPE, (PrimitiveSchema) getSchema(BasicOntology.STRING));
      cs.add(CREDIT_CARD_NUMBER, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
      cs.add(CREDIT_CARD_EXPIRATION_DATE, (PrimitiveSchema) getSchema(BasicOntology.DATE), ObjectSchema.OPTIONAL);

      AgentActionSchema as = (AgentActionSchema) getSchema(SELL);
      as.add(SELL_BUYER, (ConceptSchema) getSchema(BasicOntology.AID));
      as.add(SELL_ITEM, (ConceptSchema) getSchema(ITEM));
      as.add(SELL_CREDIT_CARD, (ConceptSchema) getSchema(CREDIT_CARD));

      PredicateSchema ps = (PredicateSchema) getSchema(OWNS);
      ps.add(OWNS_OWNER, (ConceptSchema) getSchema(BasicOntology.AID));
      ps.add(OWNS_ITEM, (ConceptSchema) getSchema(ITEM));

      ps = (PredicateSchema) getSchema(COSTS);
      ps.add(COSTS_ITEM, (ConceptSchema) getSchema(ITEM));
      ps.add(COSTS_PRICE, (ConceptSchema) getSchema(PRICE));

      useConceptSlotsAsFunctions();
    }
    catch (OntologyException oe) {
      oe.printStackTrace();
    }
  }

}
```

Ontology = 57 LOC

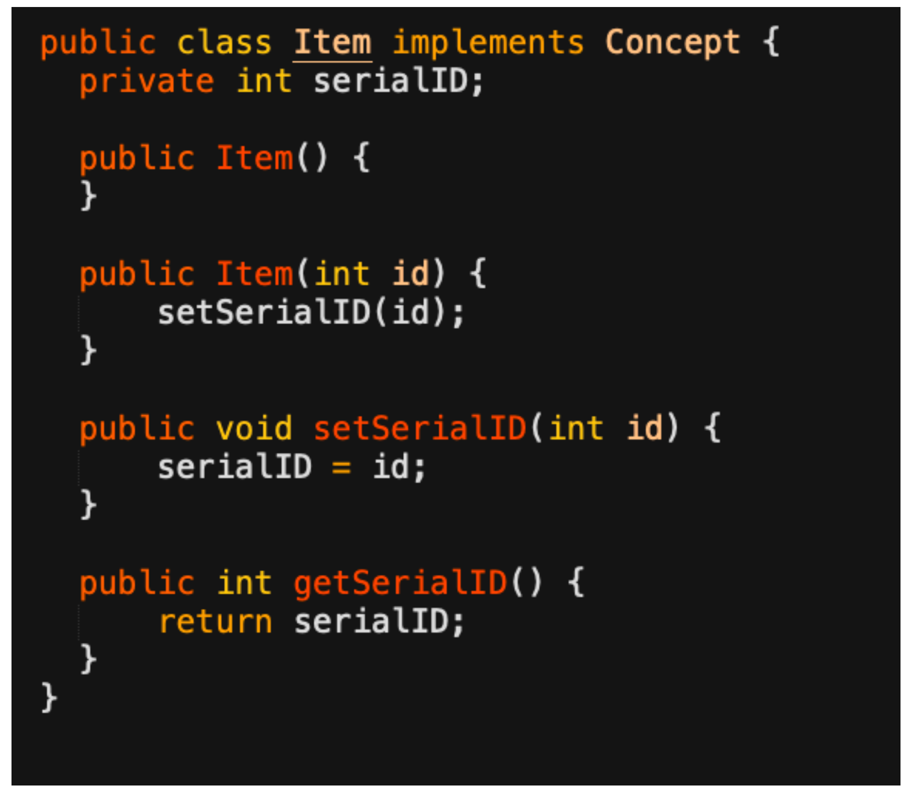
Example: JADE E-commerce Ontology (2)



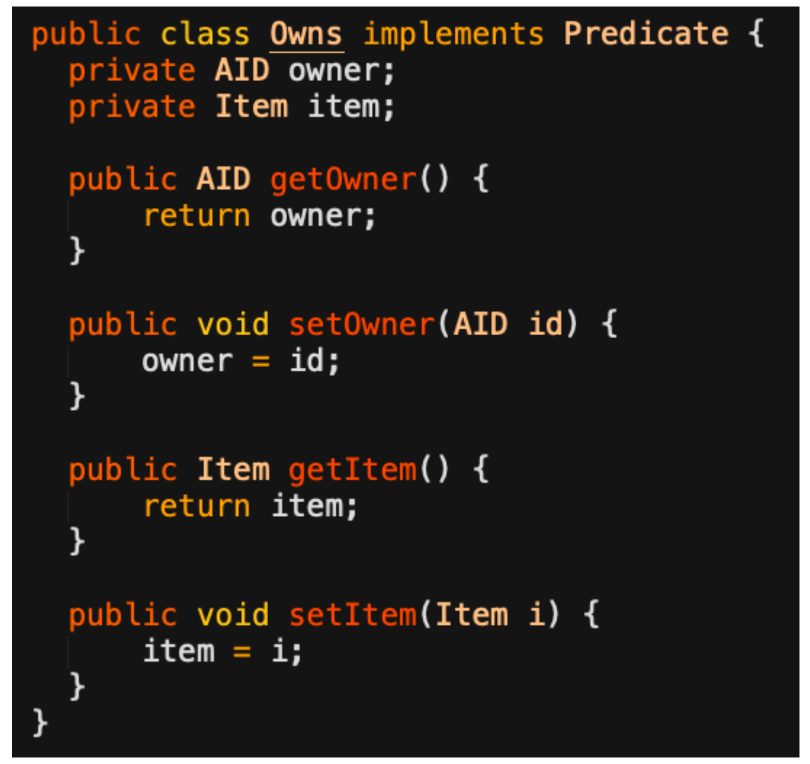
```java
public class CreditCard implements Concept {
  private String type = null;
  private long number;
  private Date expirationDate = null;

  public CreditCard() {
  }

  public CreditCard(String type, long number, Date expirationDate) {
      setType(type);
      setNumber(number);
      setExpirationDate(expirationDate);
  }


  public String getType() {
      return type;
  }

  public void setType(String type) {
      this.type = type;
  }


  public long getNumber() {
      return number;
  }

  public void setNumber(long number) {
      this.number = number;
  }


  public Date getExpirationDate() {
      return expirationDate;
  }

  public void setExpirationDate(Date expirationDate) {
      this.expirationDate = expirationDate;
  }

  public String toString() {
      return type+" N. "+number+" Exp. "+expirationDate;
  }
}
```



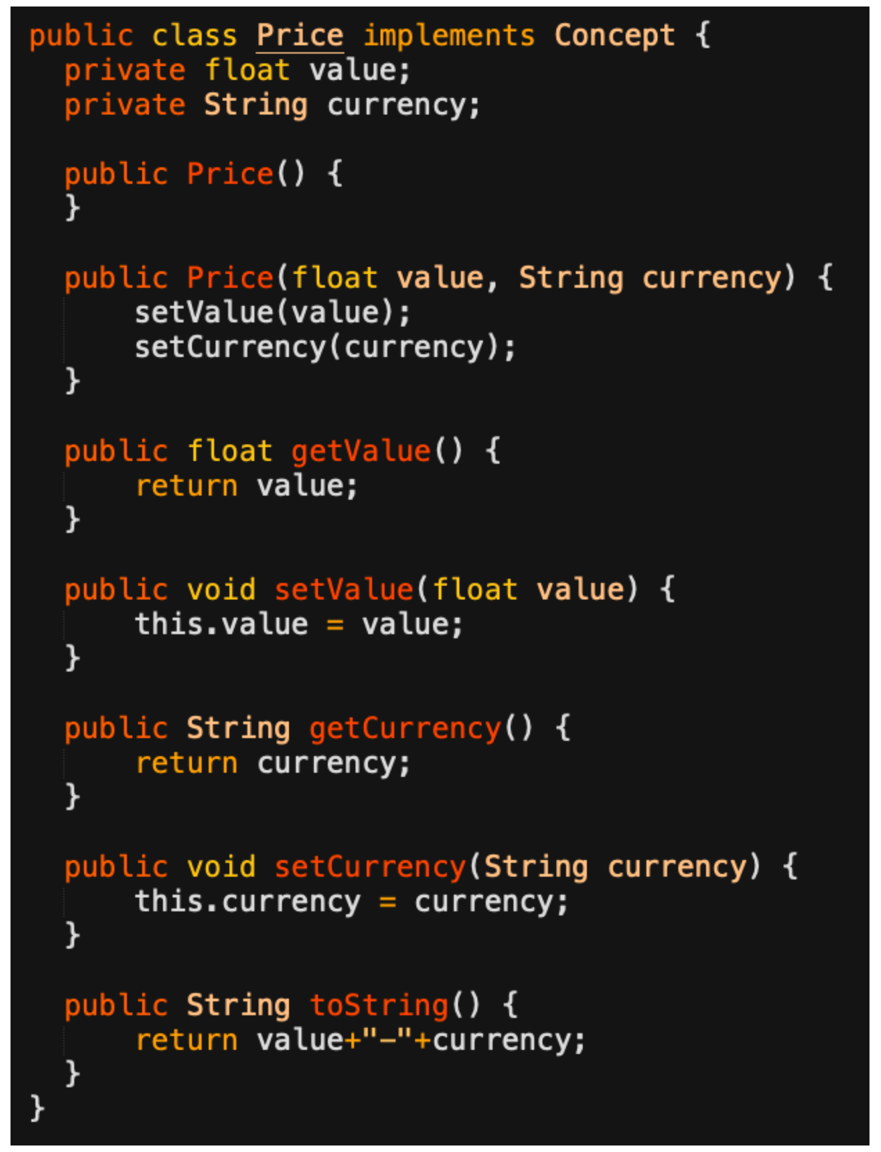
```java
public class Costs implements Predicate {
  private Item item;
  private Price price;

  public Item getItem() {
      return item;
  }

  public void setItem(Item i) {
      item = i;
  }

  public Price getPrice() {
      return price;
  }

  public void setPrice(Price p) {
      price = p;
  }

}
```



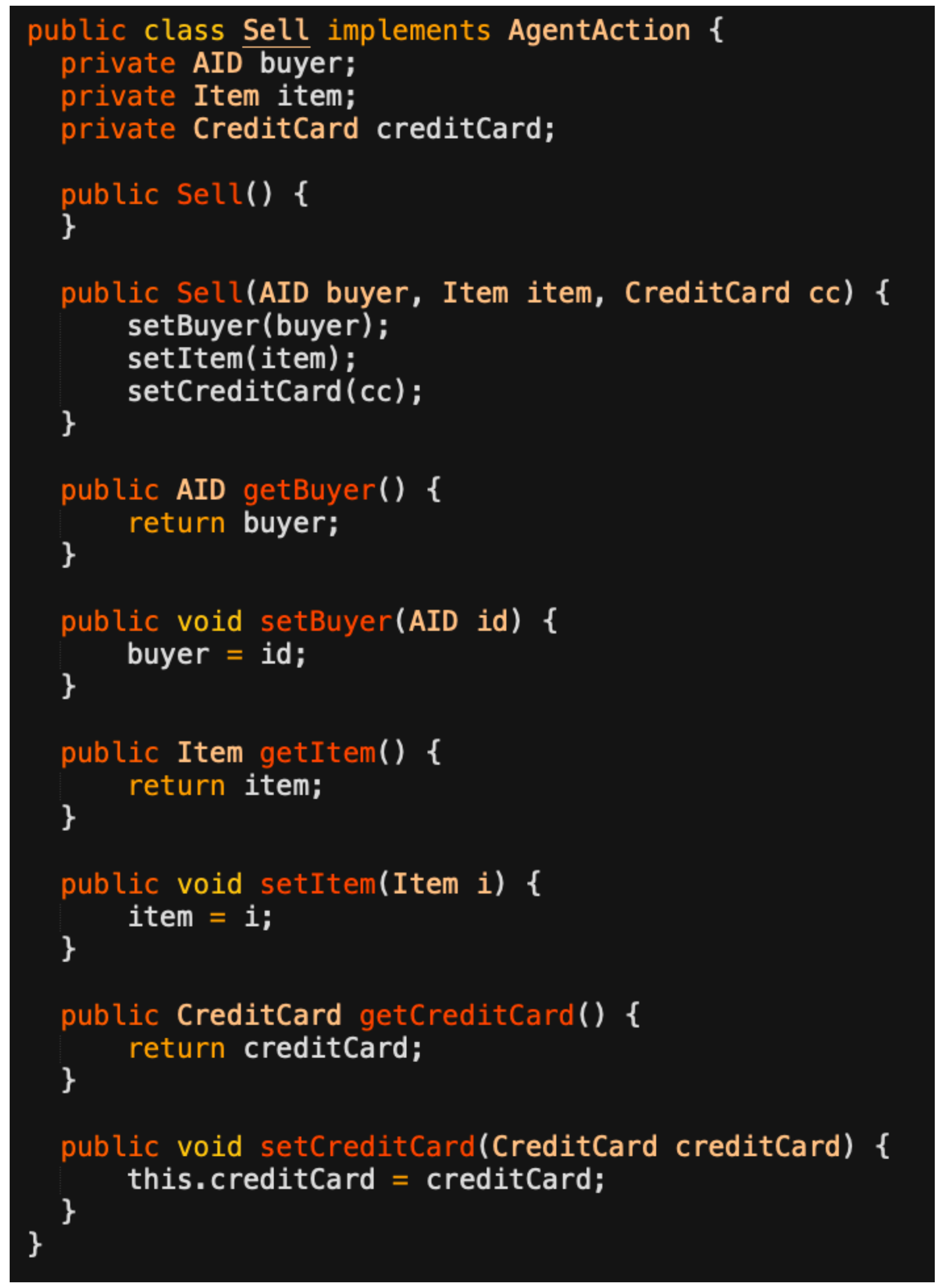
```java
public class Item implements Concept {
  private int serialID;

  public Item() {
  }

  public Item(int id) {
      setSerialID(id);
  }

  public void setSerialID(int id) {
      serialID = id;
  }

  public int getSerialID() {
      return serialID;
  }
}
```



```java
public class Owns implements Predicate {
  private AID owner;
  private Item item;

  public AID getOwner() {
      return owner;
  }

  public void setOwner(AID id) {
      owner = id;
  }

  public Item getItem() {
      return item;
  }

  public void setItem(Item i) {
      item = i;
  }
}
```



```java
public class Price implements Concept {
  private float value;
  private String currency;

  public Price() {
  }

  public Price(float value, String currency) {
      setValue(value);
      setCurrency(currency);
  }

  public float getValue() {
      return value;
  }

  public void setValue(float value) {
      this.value = value;
  }

  public String getCurrency() {
      return currency;
  }

  public void setCurrency(String currency) {
      this.currency = currency;
  }

  public String toString() {
      return value+"-"+currency;
  }
}
```



```java
public class Sell implements AgentAction {
  private AID buyer;
  private Item item;
  private CreditCard creditCard;

  public Sell() {
  }

  public Sell(AID buyer, Item item, CreditCard cc) {
      setBuyer(buyer);
      setItem(item);
      setCreditCard(cc);
  }

  public AID getBuyer() {
      return buyer;
  }

  public void setBuyer(AID id) {
      buyer = id;
  }

  public Item getItem() {
      return item;
  }

  public void setItem(Item i) {
      item = i;
  }

  public CreditCard getCreditCard() {
      return creditCard;
  }

  public void setCreditCard(CreditCard creditCard) {
      this.creditCard = creditCard;
  }
}
```

= 175 LOC

Example: Jadescript E-commerce Ontology

```
ontology ECommerce
    concept Item(serialID as integer)
    concept CreditCard(type as text, number as integer,
        expirationDate as timestamp)
    concept Price(value as real, currency as text)

    action Sell(buyer as aid, item as Item, creditCard as CreditCard)

    predicate Owns(owner as aid, item as Item)
    predicate Costs(item as Item, price as Price)
```

= 10 LOC (vs 258 Java+JADE LOC)

Jadescript: Agents

- Like JADE agents
- include:
    - lifecycle event handlers
        - on create, on destroy
    - functions, procedures
    - properties

```
agent ECommerceSeller
   uses ontology ECommerce

   property catalogue as list of Price

   on create do
      activate ShopKeeping
```

Jadescript: Behaviours

- Based on JADE simple behaviours (**cyclic, one shot**)
- include:
    - lifecycle event handlers
        - on create, on destroy
        - **on activate, on deactivate**
    - **message event handlers**
        - **with declarative pattern matching!**
    - generic action entry points (**on execute**)
    - functions, procedures
    - properties
- can be activated with a delay and a period

```
behaviour ShopKeeping
  uses ontology ECommerce
  for agent ECommerceSeller

  on message request Sell(b, i, c) do
    if cardValid(c) do
      send message agree Sell(b, i, c) to sender
      activate ProcessPayment(priceOf(i), i, b, c)
    else do
      send message refuse Sell(b, i, c) to sender
```

Jadescript: Opportunities for the future

Opportunities to add other abstractions, e.g.:

- Practical reasoning agents, Belief-Desire-Intention (JADEX)
    - Ontologies to also incorporate reasoning aspects (e.g., rules)?
    - Behaviours as plans? plans as behaviours?
- User-defined interaction protocols

References

- Wooldridge, M. (2009). An Introduction to MultiAgent Systems - 2nd Edition. John Wiley & Sons.
- Franklin, S. & Graesser, A. (1996). Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents.
- Bergenti, F., Caire, G., Monica, S., & Poggi, A. (2020). The First Twenty Years of Agent-Based Software Development with JADE. Autonomous Agents and Multi-Agent Systems. Springer.
- Bellifemine, F., Caire, G., & Greenwood, D. (2007). Developing Multi-Agent Systems with JADE. John Wiley & Sons.
- Pokahr, A., Braubach, L., & Lamersdorf, W. (2005). Jadex: A BDI reasoning engine. In Multi-agent Programming. Springer.
- Shoham, Y. (1993). Agent-oriented programming. Artificial Intelligence.
- Petrosino, G., & Bergenti, F. (2018). An Introduction to the Major Features of a Scripting Language for JADE Agents. Lecture Notes in Computer Science, LNAI
- Petrosino, G., & Bergenti, F. (2019). Extending Message Handlers with Pattern Matching in the Jadescript Programming Language. CEUR Workshop Proceedings.

FAQs: Isn't it all just Distributed Systems?

- Distributed Systems already tackle challenges brought by ubiquity and interconnectivity of computing systems
- However, in DS the components usually do not require two aspects:
    - the ability to act independently/autonomously
    - the ability to act in a self-interested way

FAQs: Isn't it all just Artificial Intelligence?

- AI has focused on *components* of intelligence
  - planning, learning, searching, logical reasoning...
- Agents *can* integrate such components
  - ... they might not: standard CS and good Software Engineering could be enough

- AI tends to ignore *social* aspects of intelligence

- intelligence can also emerge from **societies**

FAQs: Isn't it all just Economics/Game Theory?

- Both fields provide points of inspiration for MAS, but
  - tend to provide *descriptive* concepts, telling the properties of an appropriate/optimal solution without telling us *how* to compute the solution
  - often the solutions do not take into account that systems that operate in the real world are resource-bounded
  - some assumptions (e.g., the notion of rational agents) may not be valid or useful for engineering artificial agents

FAQs: Isn't it all just Social Sciences?

- MAS could take inspiration from the way human and other natural societies work
    - (...the other way around is also true!)
- However, it does not follow we should build artificial societies in the same way
    - natural societies could be not the best societies
    - it is hard to model precisely the behaviour of human societies

FAQs: Isn't an agent just a program?

- Not all programs are agents
    - (Franklin, Graesser): "*An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it,* **over time***, in pursuit of its own agenda and so as* **to effect what it senses in the future**."
    - e.g., `ls` is not an agent
        - it senses and acts through its inputs and outputs
        - but has no time continuity; it does not perceive what has done before
- Not all agents are programs
    - e.g., a bimetal thermostat, a robot, a school teacher, a company