SYS466 Analysis and Design

Lecture 6 - Object Level Behavioural Modelling
School of Information and Communications Technology
Seneca College

Models

Use Case/Analysis Modelling

- use case diagrams
- use case descriptions/ scenarios
- system level sequence diagrams
- activity diagrams

Domain Analysis

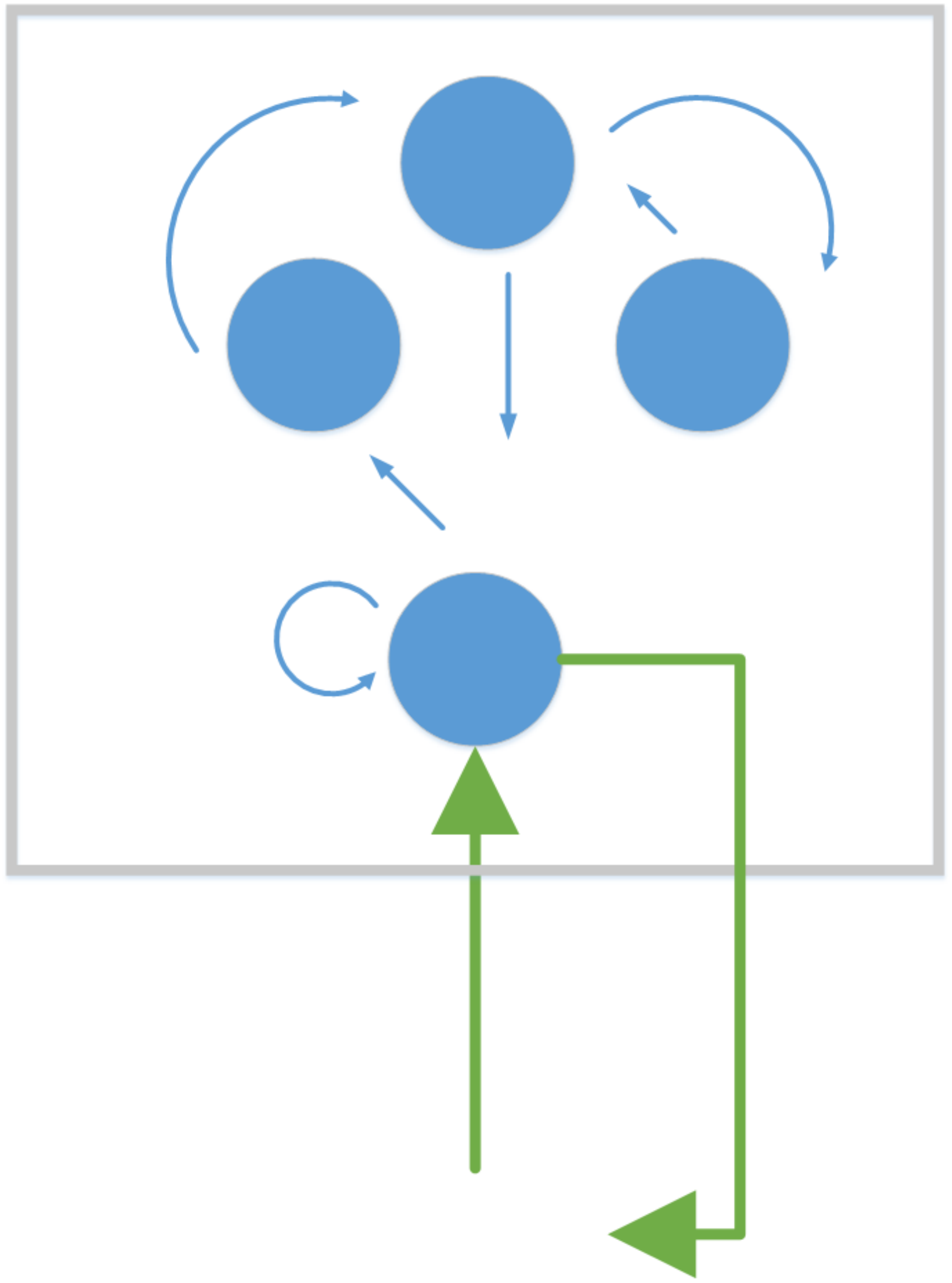
- initial conceptual class definitions
- attributes
- relationships/multiplicity

Design Modelling

- design level classes
- object level sequence diagrams
- object level activity diagrams

Object Collaboration

- occurs when an object can't fulfil its responsibility alone
- in most nontrivial systems, a service is provided by a group of objects
- objects tend to focus on a <u>single concern</u>
- objects can participate in multiple ones simultaneously



Collaboration Example

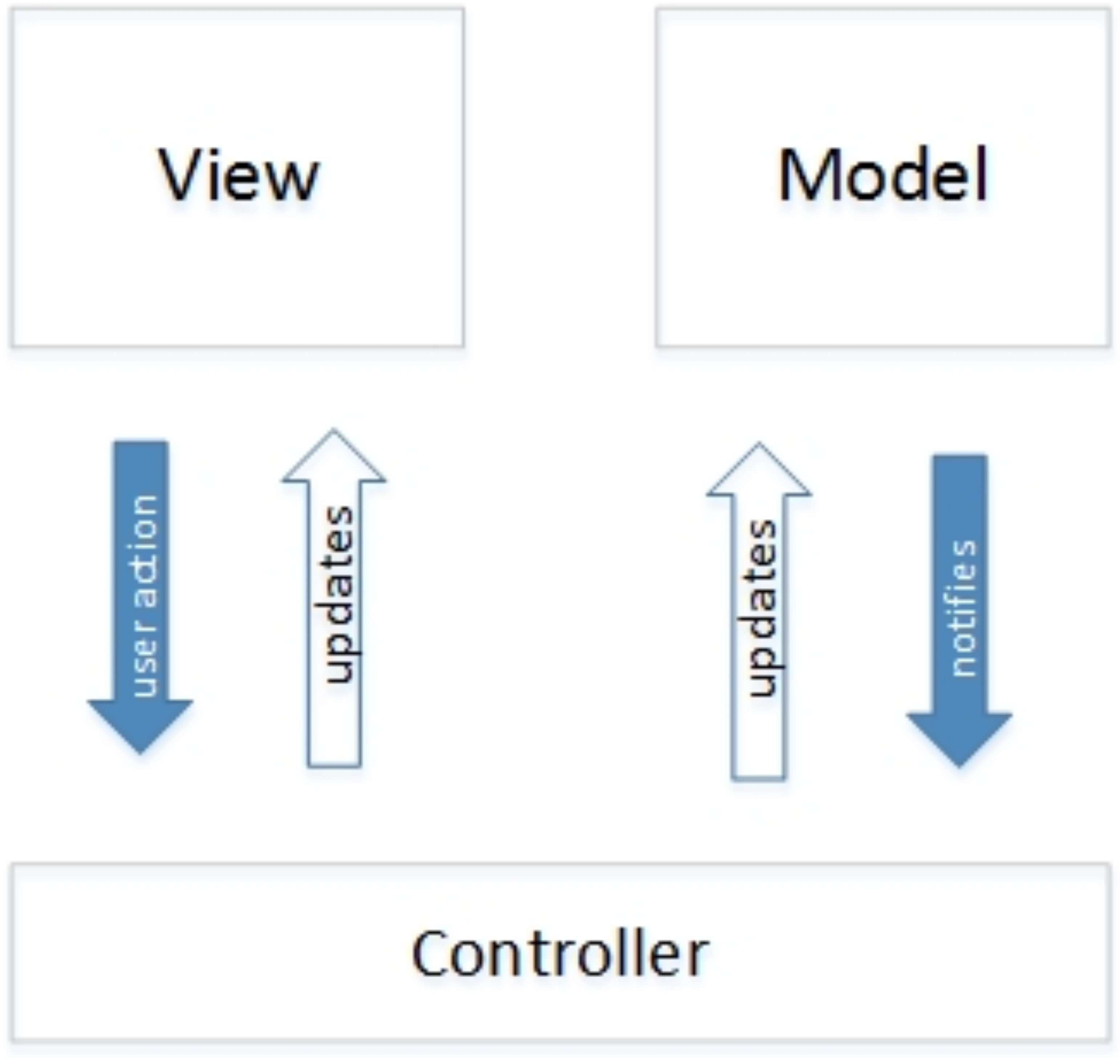
```
class Store { Employee e; Warehouse w; ….}

Product* Store::sellProduct(int productID) {
  if (w.isProductAvailable(productID) and e.isAuthorizedSeller() )
     Product p = w.removeProductFromInventory(productID);
     p.setSoldBy(e);
     return p;
  }}

main() {
  Store s = new Store();
  …
  s.signIn(new Employee(“Jack”) );
  s.addWarehouse(new Warehouse(“Toronto Depot”) );
  …
  Product p = s.sellProduct(345);}
```

Model View Controller Pattern

- system is defined by three distinct roles
- view, responsible of user presentations
- model, responsible for encapsulating data
- controller, acts as intermediary between view and model

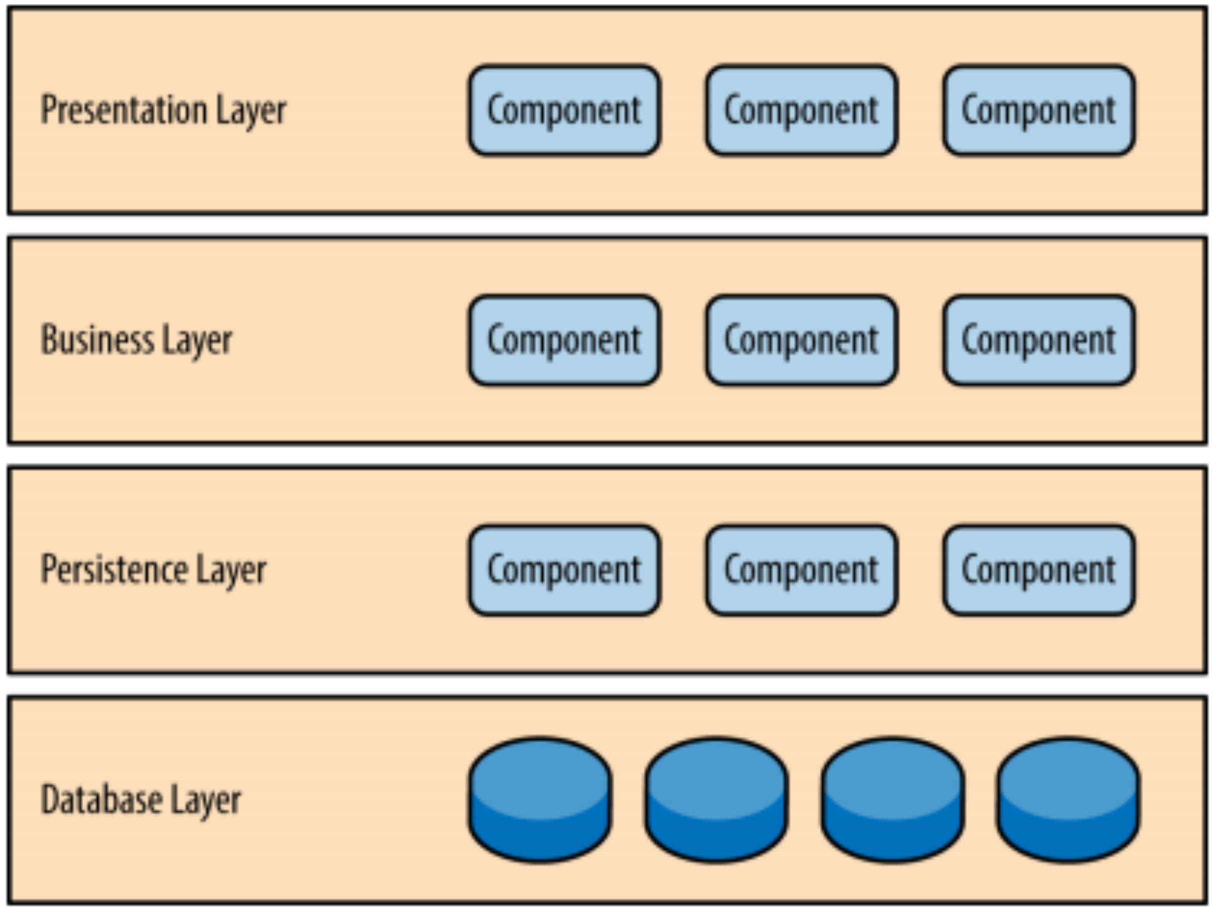


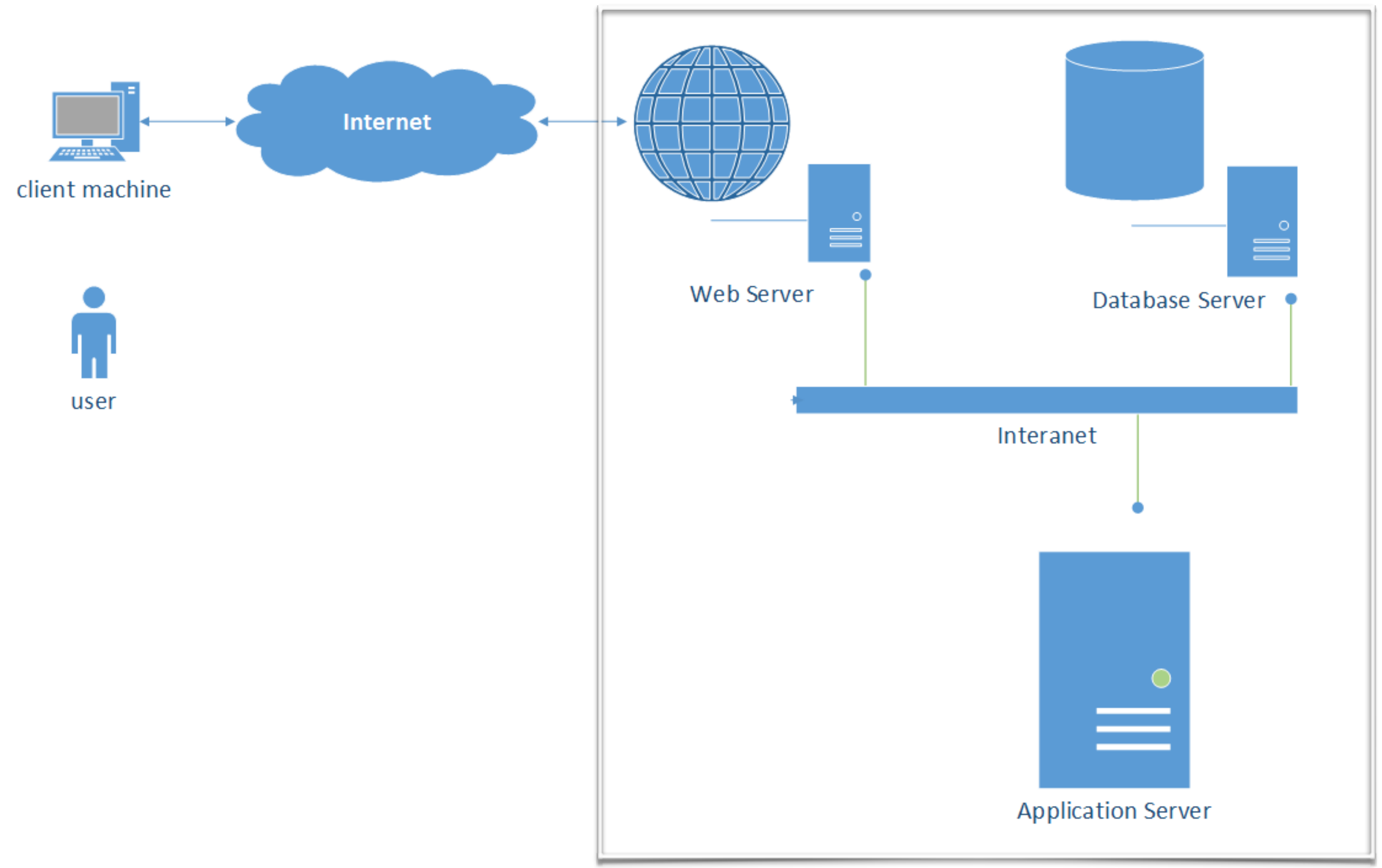
Layering Pattern Revisited

- collect components that are highly dependent on each other into logical group (highly cohesive)

- group created should ideally not be dependent on other components (highly decoupled)

- horizontal layers are organized such that lower levels are not dependent on higher ones

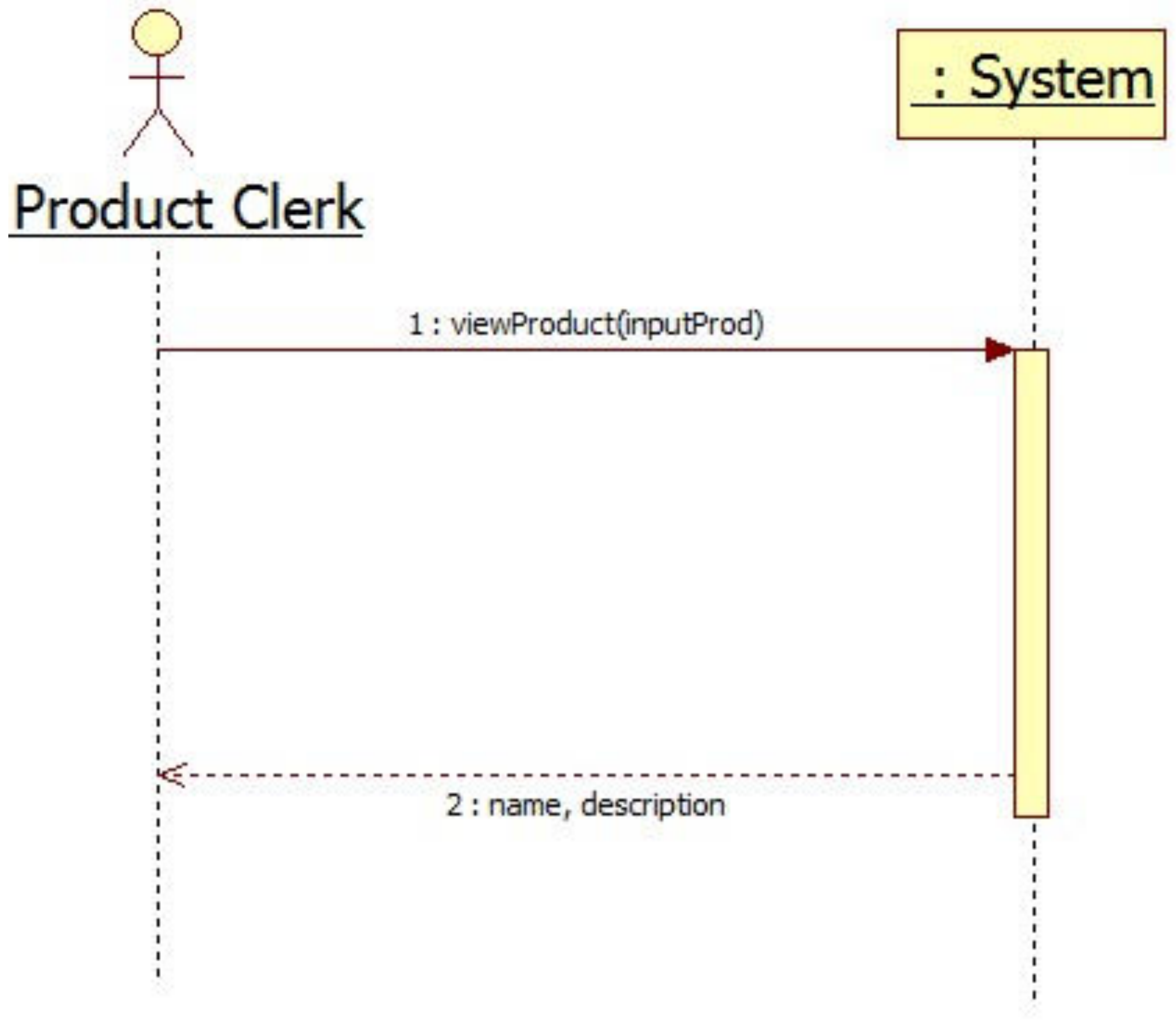




Generic IT Architecture

1. Web Server - render ui for client's browser
2. Application Server - provides business logic to perform services
3. Database Server - stores system data

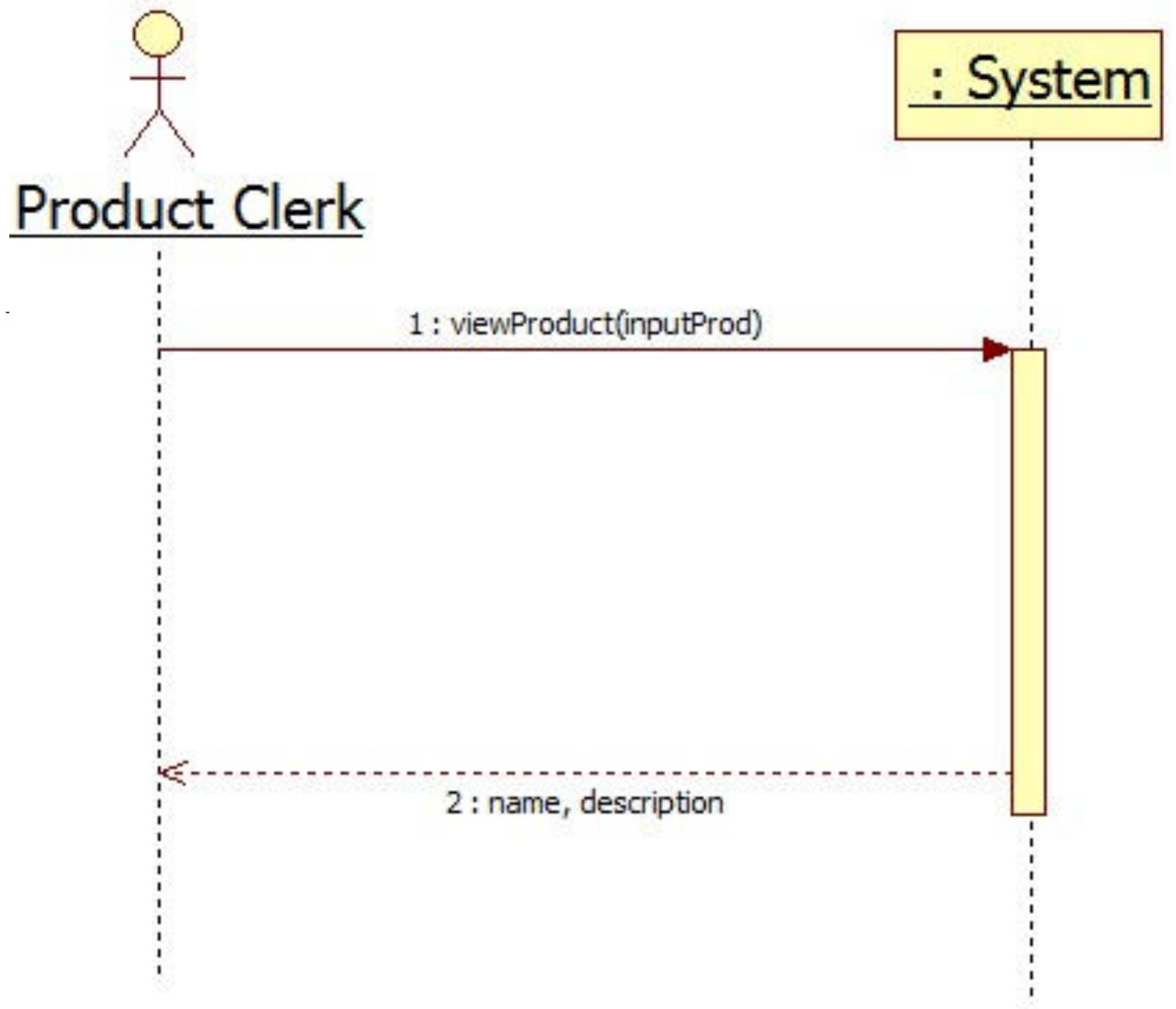
System Sequence Diagrams



- actor's point of view
- part of use case model
- system is a "black box"

Object System Sequence Diagrams

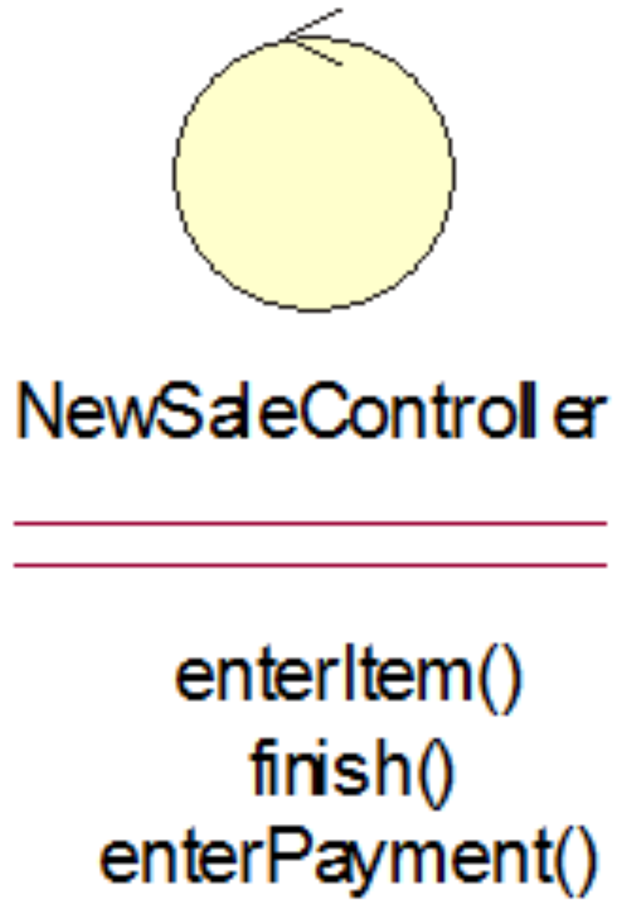
- opens up the "black box"
- documents interaction for a single <u>scenario</u>
- shows how objects collaborate to fulfil a request
- used to illustrate <u>ordered</u> sequence of messages between objects
- not good at describing exact behaviour



System Details

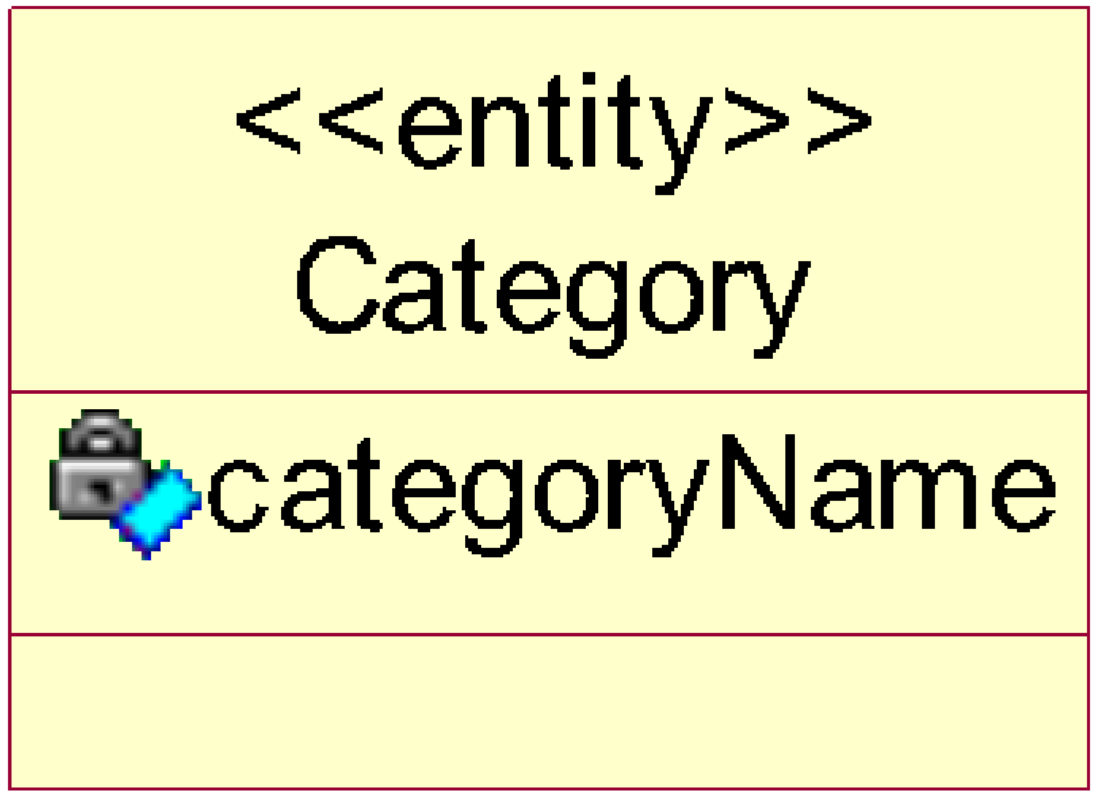
Controller Classes

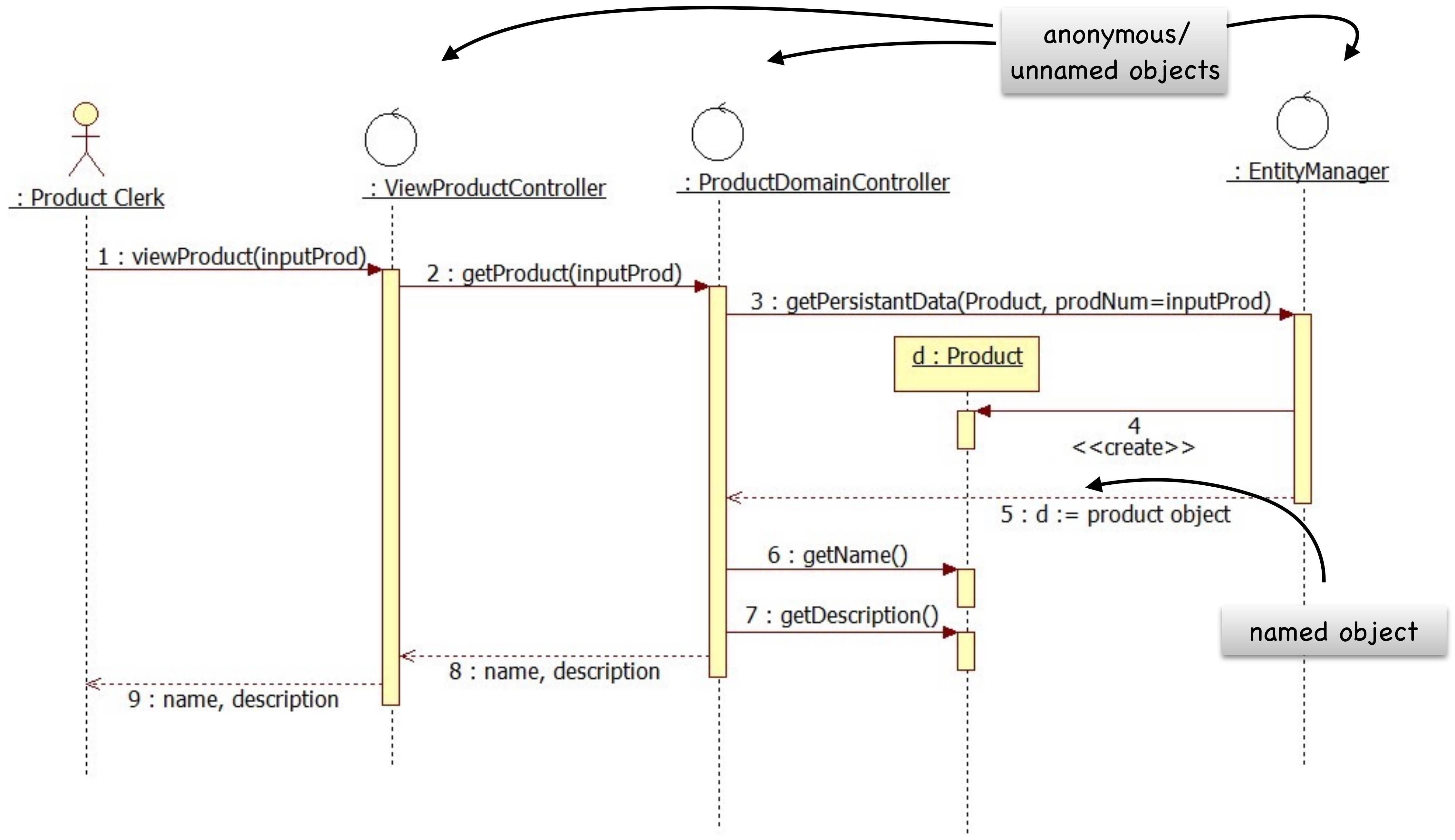
- control/coordinate the system behaviour
- delegates work to other classes
- decouples layers



Entity Classes

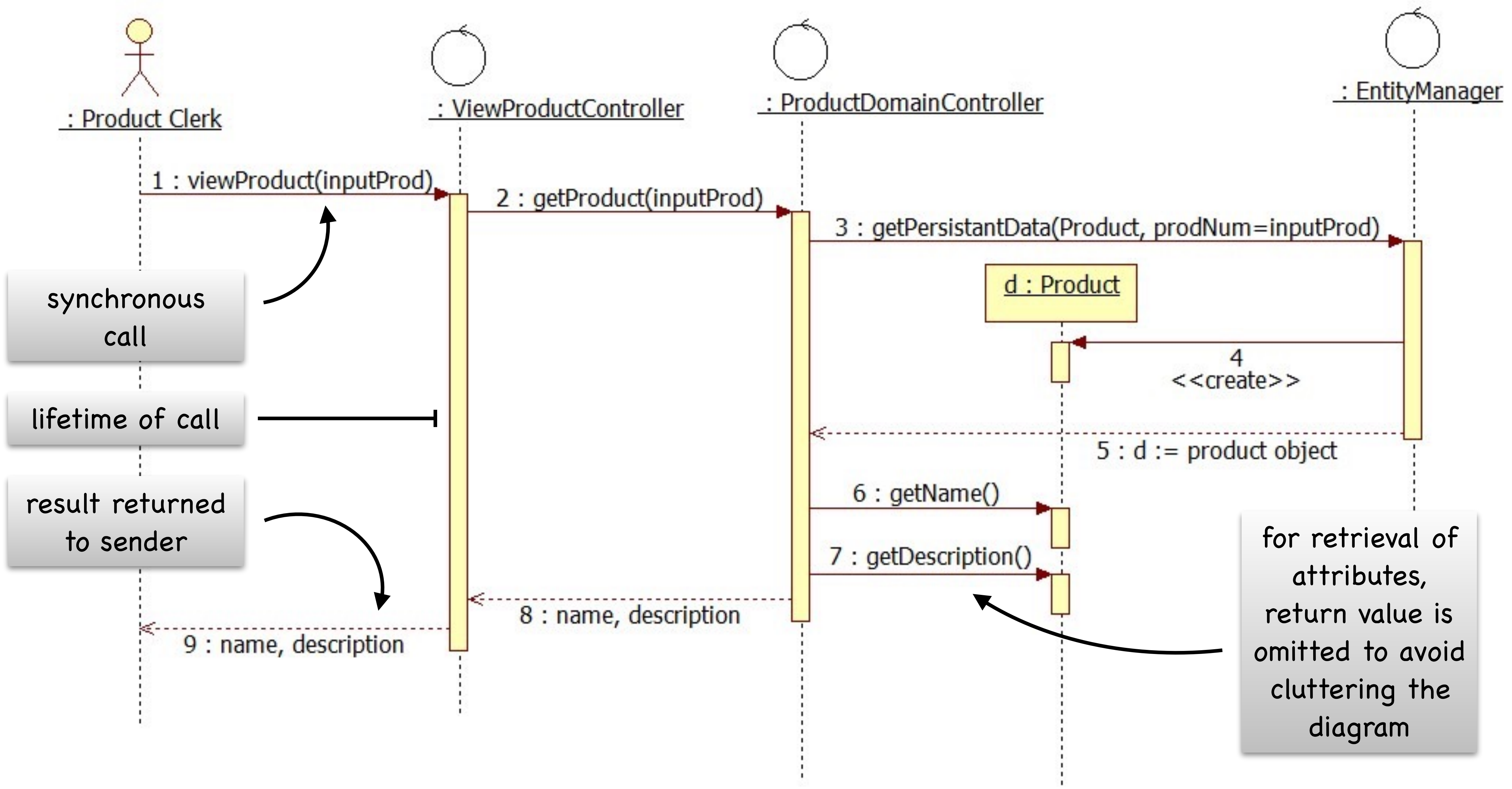
- data classes
- model data in the system
- often map directly to conceptual classes in the domain model
- in this course, we leave out the <<entity>> as per convention





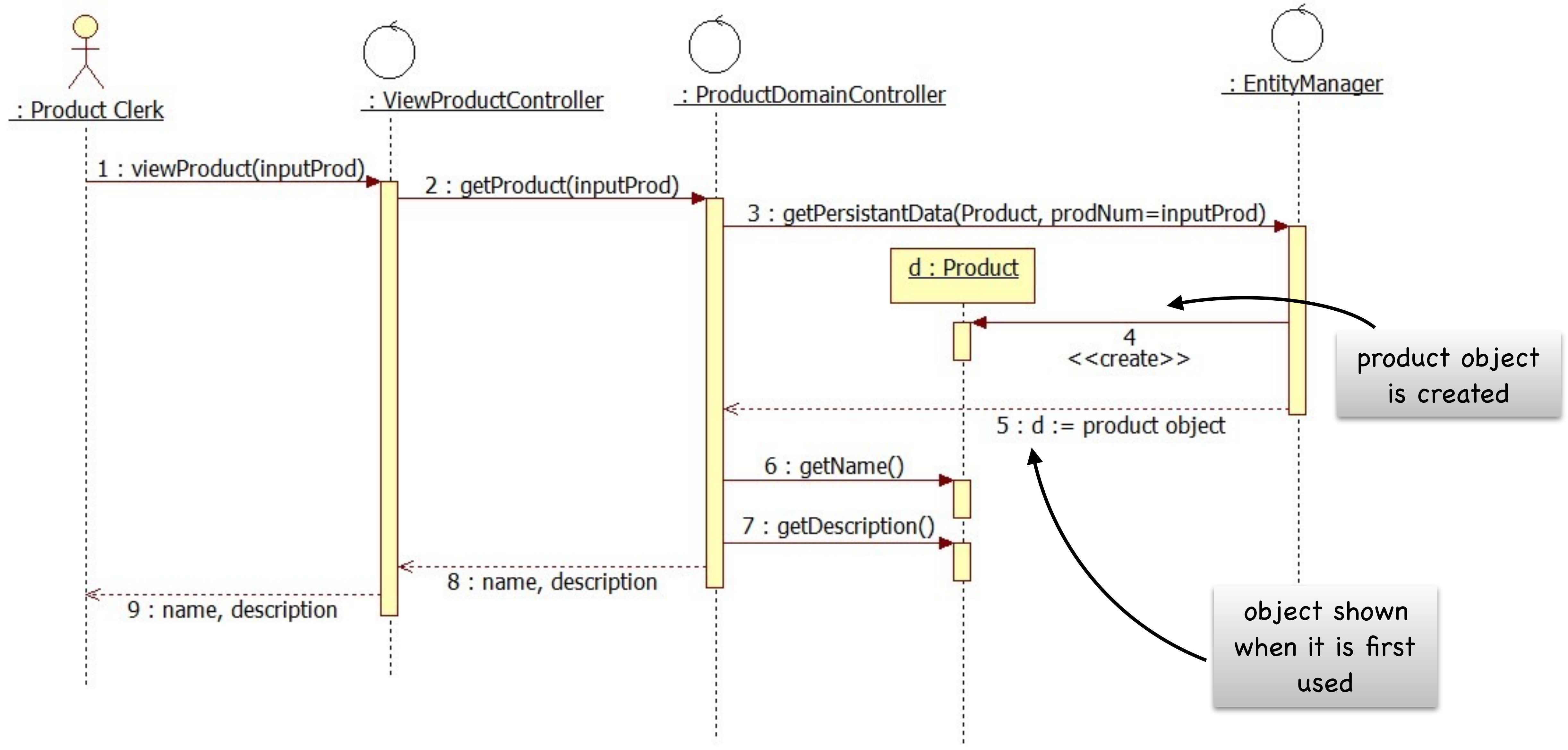
Object Naming

Objects are defined by *objectIdentifer:classIdentifier* notation


: Product Clerk
: ViewProductController
: ProductDomainController
: EntityManager
1 : viewProduct(inputProd)
2 : getProduct(inputProd)
3 : getPersistantData(Product, prodNum=inputProd)
d : Product
4
<<create>>
5 : d := product object
6 : getName()
7 : getDescription()
8 : name, description
9 : name, description
synchronous call
lifetime of call
result returned to sender
for retrieval of attributes, return value is omitted to avoid cluttering the diagram

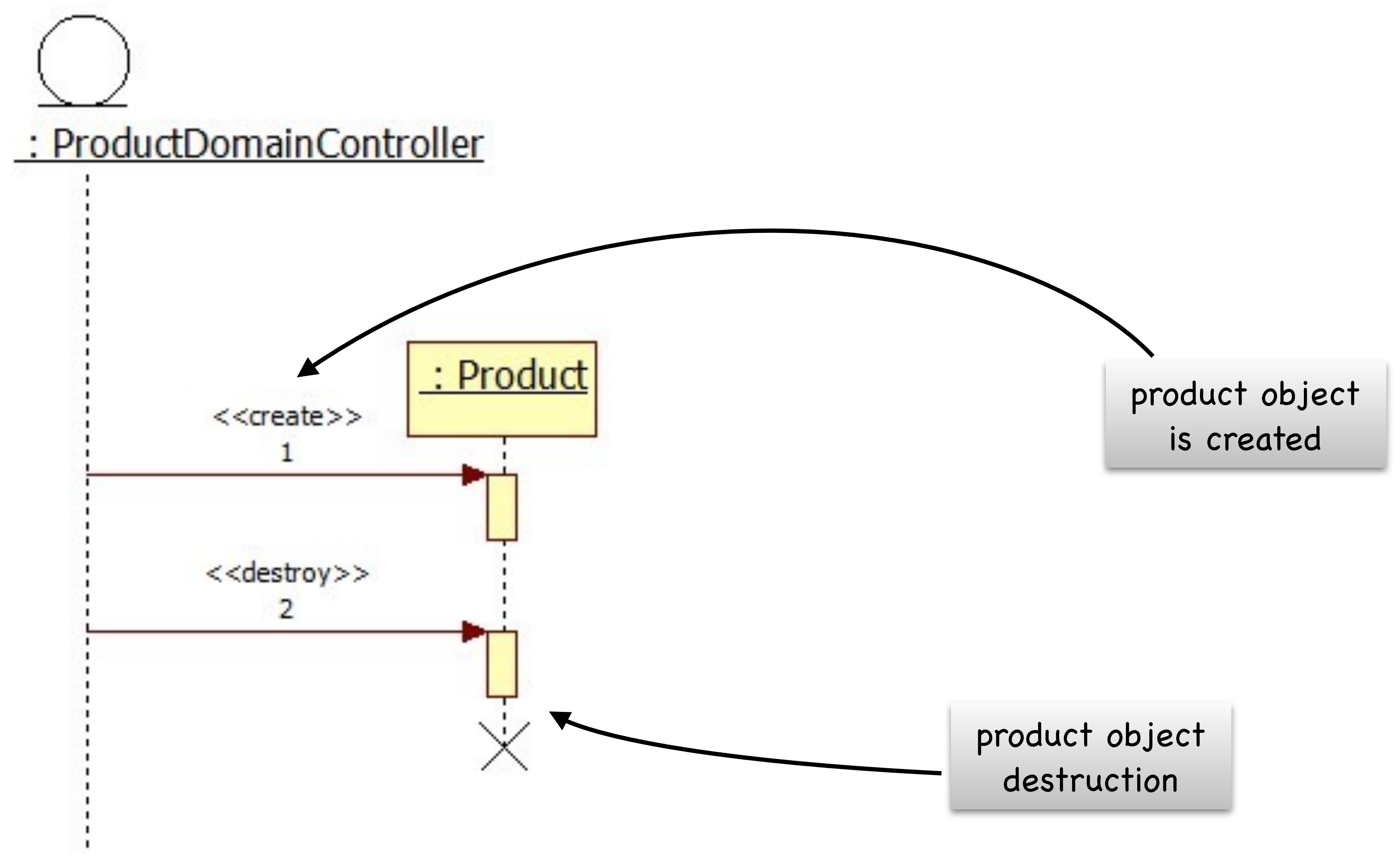
Method Calls

Models invoking a class method on an object or triggering a event


: Product Clerk
: ViewProductController
: ProductDomainController
: EntityManager
1 : viewProduct(inputProd)
2 : getProduct(inputProd)
3 : getPersistantData(Product, prodNum=inputProd)
d : Product
4
<<create>>
product object is created
5 : d := product object
6 : getName()
7 : getDescription()
8 : name, description
9 : name, description
object shown when it is first used

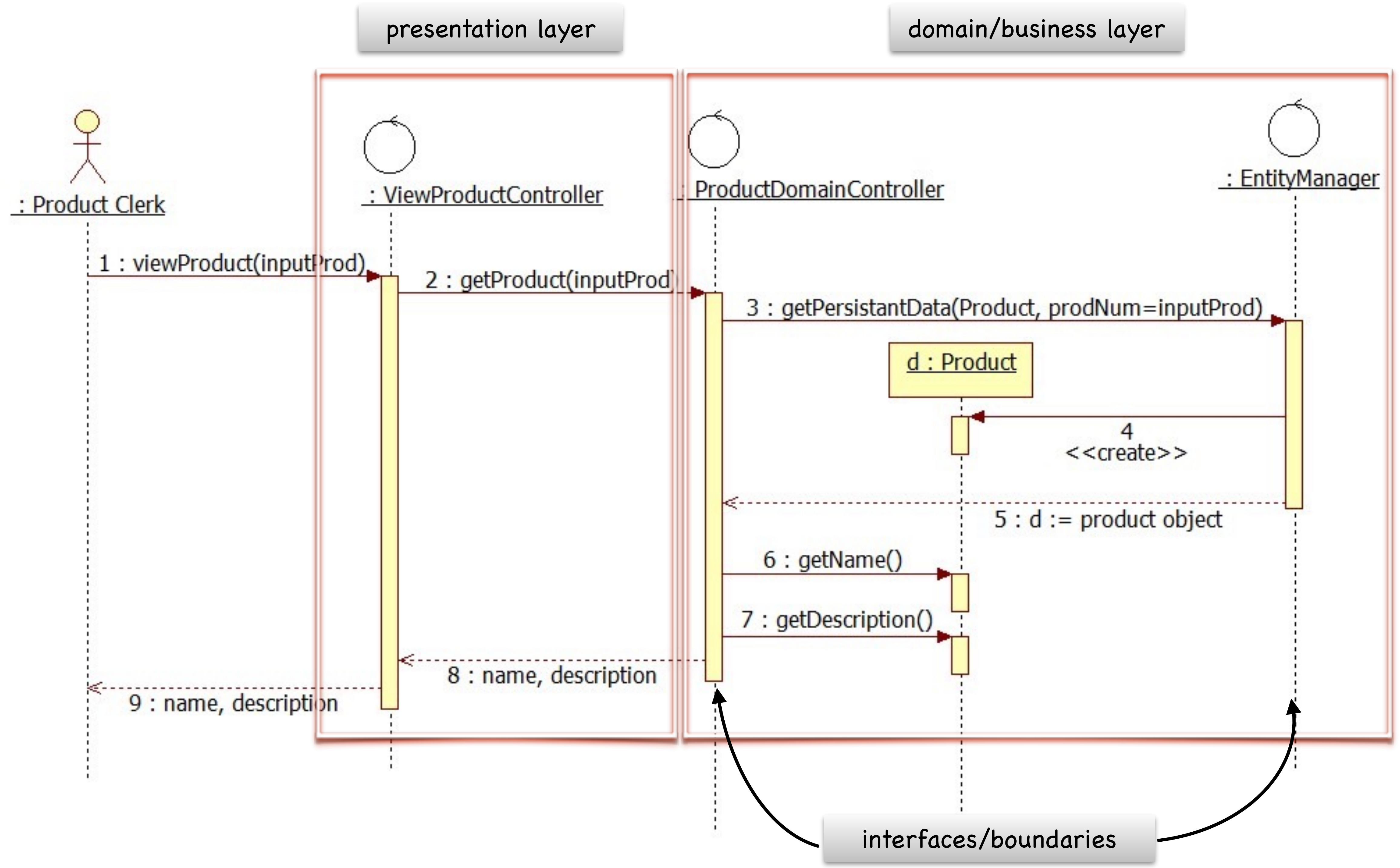
Lifelines

indicates when an object exists in scenario


: ProductDomainController
: Product
<<create>>
1
<<destroy>>
2
product object is created
product object destruction

Lifelines

<<create>> and <<destroy>> stereotypes indicate object construction and destruction calls

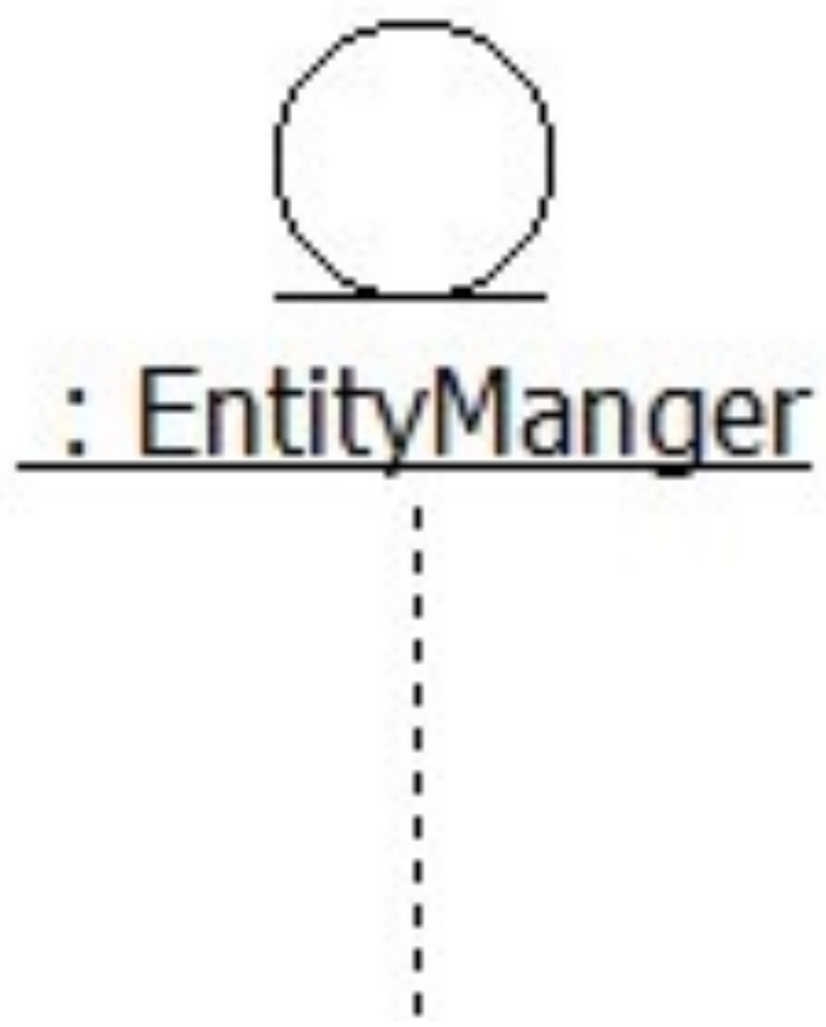


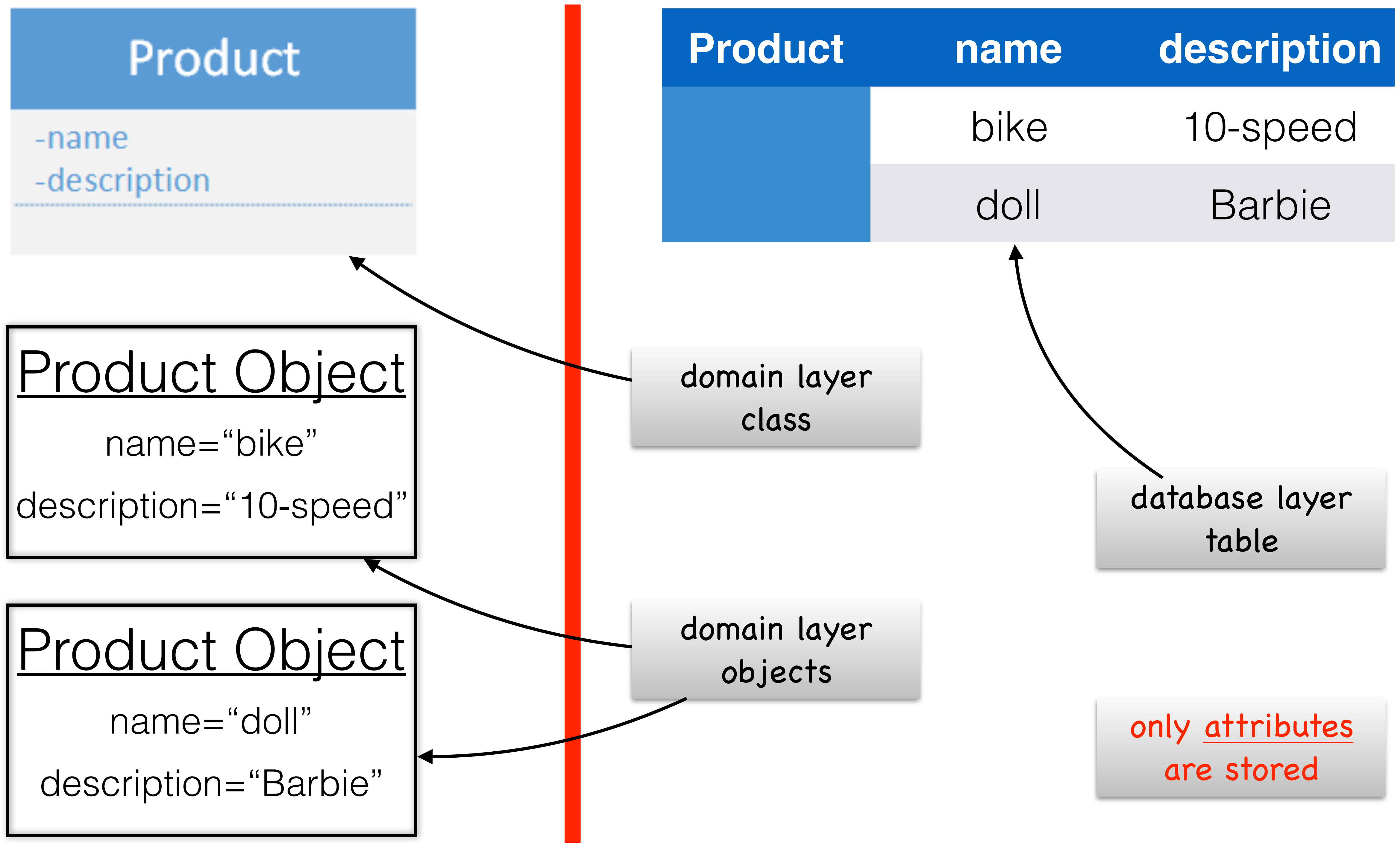
Layered Pattern Collaborations

Objects across layers collaborate to provide services
Controllers provides interfaces to access to neighbouring layers

Persistance Layer

- used to permanently store system data
- performs translation between data store and system objects
- uses unique object identifiers to ensure a mapping between system objects and representation in the data store
- for this course, access to this layer's services will be provided by standard methods in the **EntityManager** Controller class

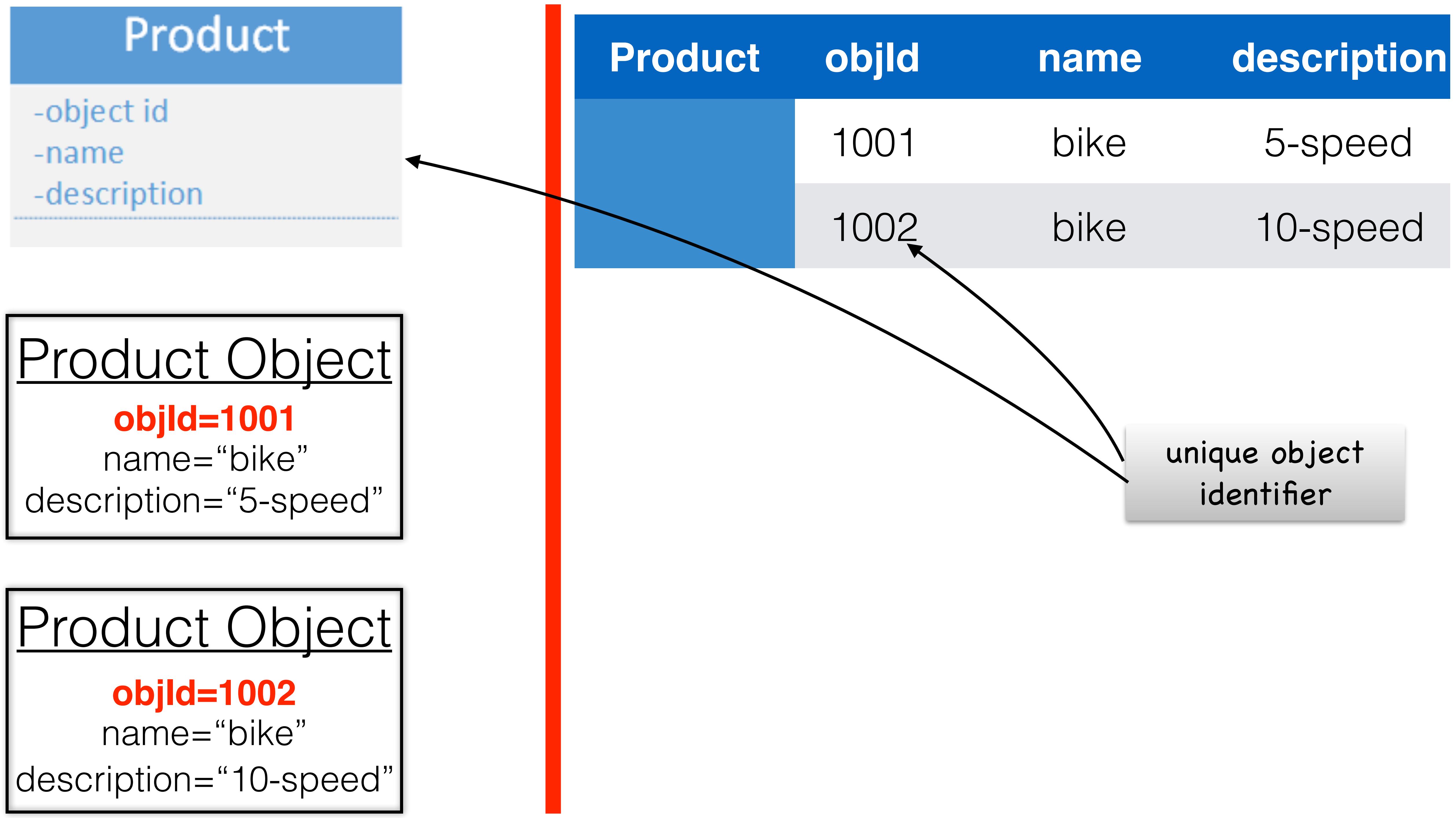




Product	name	description
	bike	10-speed
	doll	Barbie

Object as Tables Pattern

each domain object is mapped to a table
a unique table is used for every class
in relational database terminology, each object is mapped to a <u>row</u> in the table



Product	objId	name	description
	1001	bike	5-speed
	1002	bike	10-speed

Object Identifier Pattern

provides a consistent way for accessing objects
ensures objects are unique
in relational database terms, the object identifier would be a table's <u>primary key</u>

Entity Manager Services

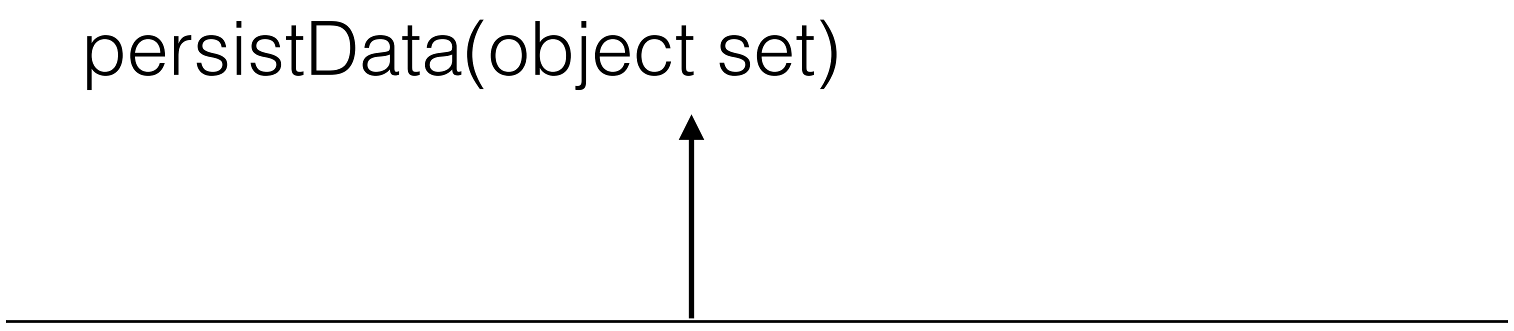
getPersistantData(ClassName, boolean expression)

entity class in the domain/business layer

some expression which states which objects should retrieved from the data store

"..retrieve entity objects in the data store that satisfy some condition.."

Entity Manager Services



set of objects to be stored
(for simplicity, class type is not specified if it can be inferred from the diagram)

"..store the provided entity objects in the persistent data store.."

Entity Manager Services

deletePersistantData(ClassName, boolean expression)

entity class in the domain/business layer

some expression which states which objects should removed from the data store

"..delete give entity objects from the data store that satisfy some condition.."