Design Patterns

Jeszenszky, Péter
University of Debrecen, Faculty of Informatics
jeszenszky.peter@inf.unideb.hu

Kocsis, Gergely (English version)
University of Debrecen, Faculty of Informatics
kocsis.gergely@inf.unideb.hu

Last modified: 06.04.2017

References

java-design-patterns: Design patterns implemented in Java
https://github.com/iluwatar/java-design-patterns

Creational patterns (GoF)

- *Abstract Factory*
- Builder
- *Factory Method*
- *Object Pool*
- *Prototype*
- *Singleton*

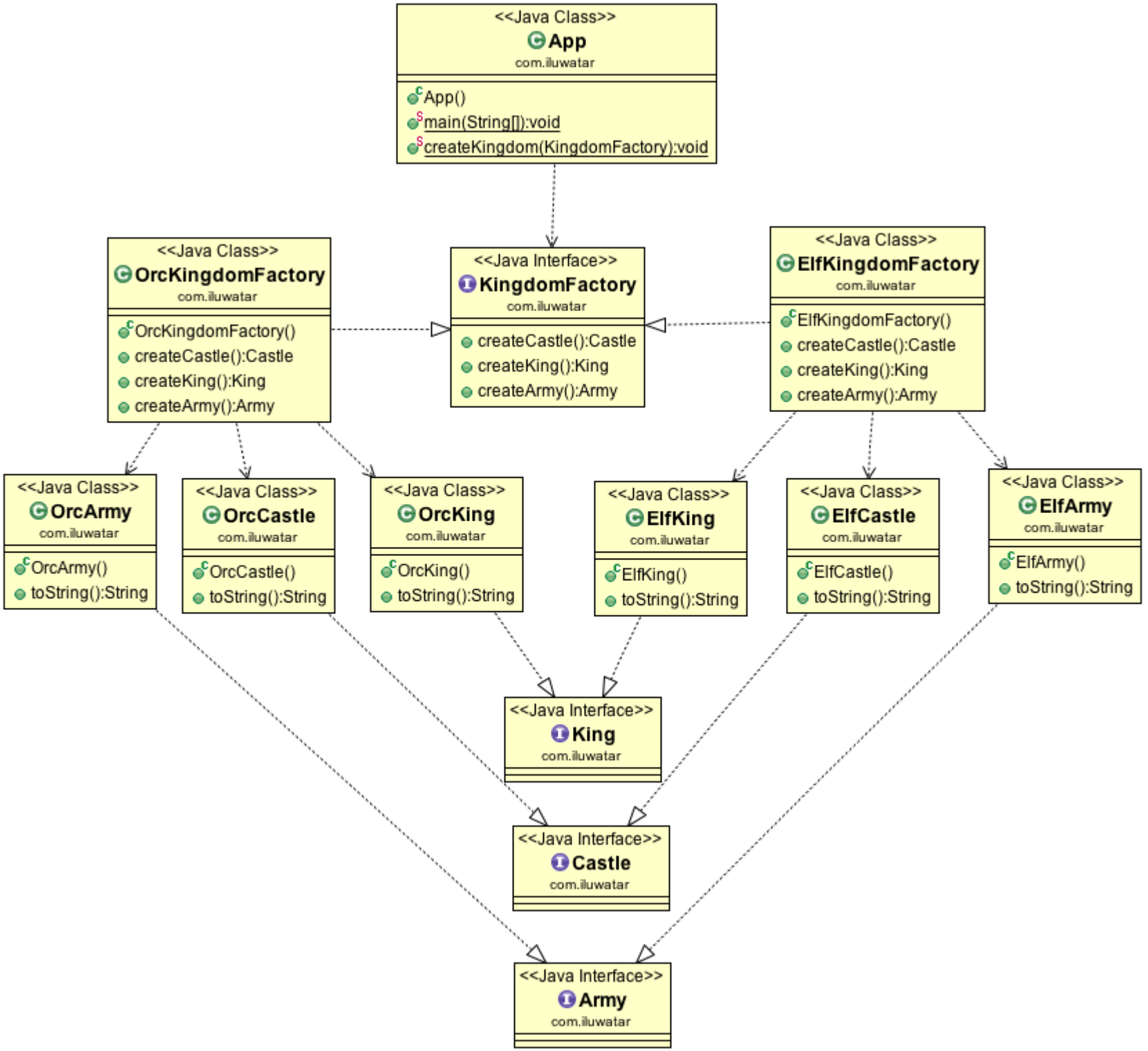
Abstract Factory (1)

Aim:

- It provides an interface to create a family of objects that are connected or depend on each other without defining the exact classes.

Abstract Factory (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/abstract-factory


<<Java Class>>
App
com.iluwatar
App()
main(String[]):void
createKingdom(KingdomFactory):void
<<Java Interface>>
KingdomFactory
com.iluwatar
createCastle():Castle
createKing():King
createArmy():Army
<<Java Class>>
OrcKingdomFactory
com.iluwatar
OrcKingdomFactory()
createCastle():Castle
createKing():King
createArmy():Army
<<Java Class>>
ElfKingdomFactory
com.iluwatar
ElfKingdomFactory()
createCastle():Castle
createKing():King
createArmy():Army
<<Java Class>>
OrcArmy
com.iluwatar
OrcArmy()
toString():String
<<Java Class>>
OrcCastle
com.iluwatar
OrcCastle()
toString():String
<<Java Class>>
OrcKing
com.iluwatar
OrcKing()
toString():String
<<Java Class>>
ElfKing
com.iluwatar
ElfKing()
toString():String
<<Java Class>>
ElfCastle
com.iluwatar
ElfCastle()
toString():String
<<Java Class>>
ElfArmy
com.iluwatar
ElfArmy()
toString():String
<<Java Interface>>
King
com.iluwatar
<<Java Interface>>
Castle
com.iluwatar
<<Java Interface>>
Army
com.iluwatar

Abstract Factory (4)

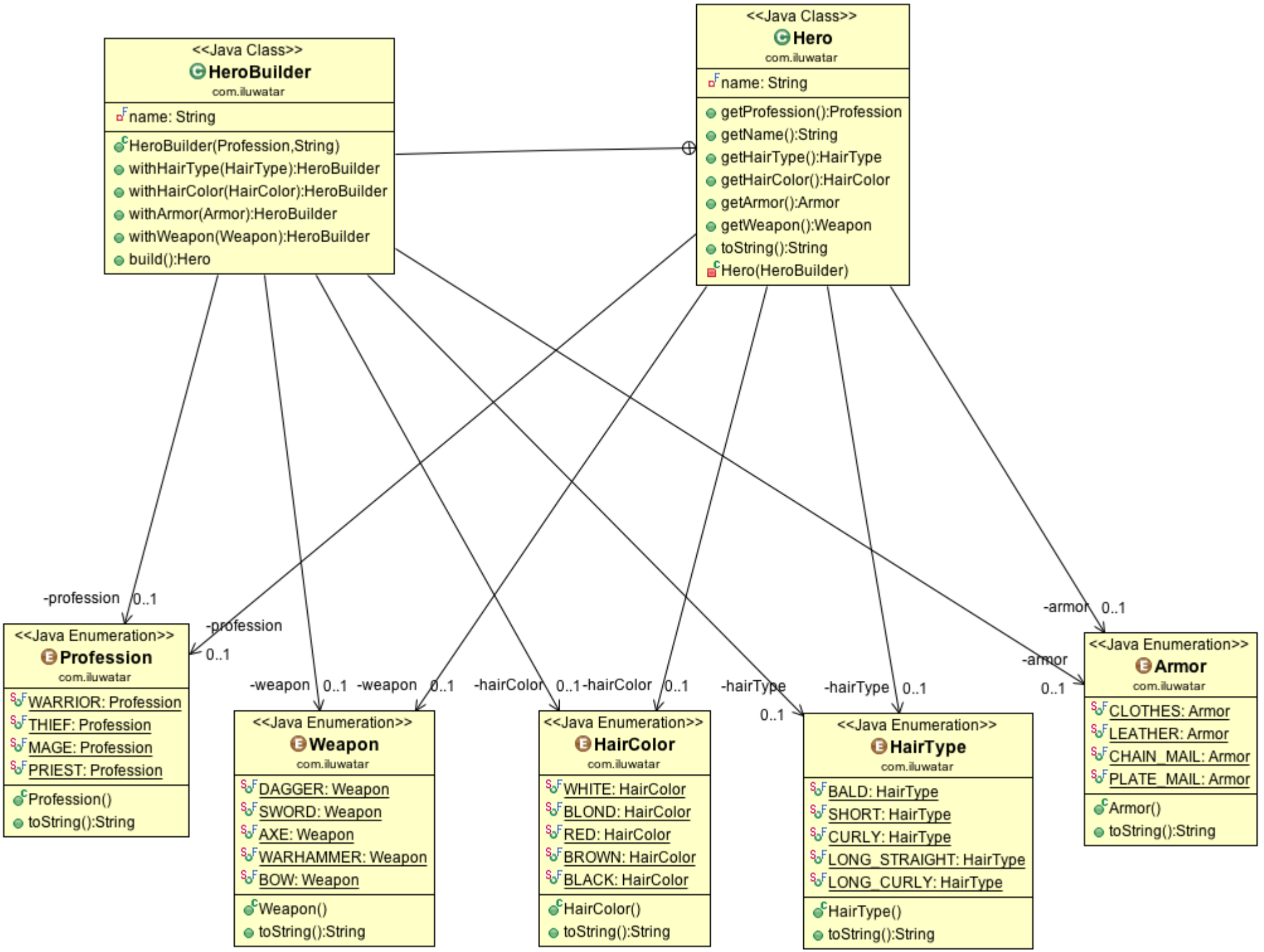
- **Known uses**
  - `java.sql.Connection` https://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html
  - `javax.xml.datatype.DatatypeFactory` https://docs.oracle.com/javase/8/docs/api/javax/xml/datatype/DatatypeFactory.html
  - `javax.xml.transform.TransformerFactory` https://docs.oracle.com/javase/8/docs/api/javax/xml/transform/TransformerFactory.html
  - `javax.xml.stream.XMLInputFactory` https://docs.oracle.com/javase/8/docs/api/javax/xml/stream/XMLInputFactory.html

Builder (1)

Aim: It makes the creation of complex objects independent of their representation. As a result with the same process we can create different representations.

Builder (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/builder


<<Java Class>>
HeroBuilder
com.iluwatar
name: String
HeroBuilder(Profession,String)
withHairType(HairType):HeroBuilder
withHairColor(HairColor):HeroBuilder
withArmor(Armor):HeroBuilder
withWeapon(Weapon):HeroBuilder
build():Hero
<<Java Class>>
Hero
com.iluwatar
name: String
getProfession():Profession
getName():String
getHairType():HairType
getHairColor():HairColor
getArmor():Armor
getWeapon():Weapon
toString():String
Hero(HeroBuilder)
-profession
0..1
-profession
0..1
-armor
0..1
-armor
0..1
-weapon
0..1
-weapon
0..1
-hairColor
0..1
-hairColor
0..1
-hairType
0..1
-hairType
0..1
<<Java Enumeration>>
Profession
com.iluwatar
WARRIOR: Profession
THIEF: Profession
MAGE: Profession
PRIEST: Profession
Profession()
toString():String
<<Java Enumeration>>
Weapon
com.iluwatar
DAGGER: Weapon
SWORD: Weapon
AXE: Weapon
WARHAMMER: Weapon
BOW: Weapon
Weapon()
toString():String
<<Java Enumeration>>
HairColor
com.iluwatar
WHITE: HairColor
BLOND: HairColor
RED: HairColor
BROWN: HairColor
BLACK: HairColor
HairColor()
toString():String
<<Java Enumeration>>
HairType
com.iluwatar
BALD: HairType
SHORT: HairType
CURLY: HairType
LONG_STRAIGHT: HairType
LONG_CURLY: HairType
HairType()
toString():String
<<Java Enumeration>>
Armor
com.iluwatar
CLOTHES: Armor
LEATHER: Armor
CHAIN_MAIL: Armor
PLATE_MAIL: Armor
Armor()
toString():String

Builder (4)

- **Known uses**:
  - `java.lang.StringBuilder` http://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html
  - `java.time.format.DateTimeFormatterBuilder` https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatterBuilder.html
  - `java.util.Locale.Builder` https://docs.oracle.com/javase/8/docs/api/java/util/Locale.Builder.html

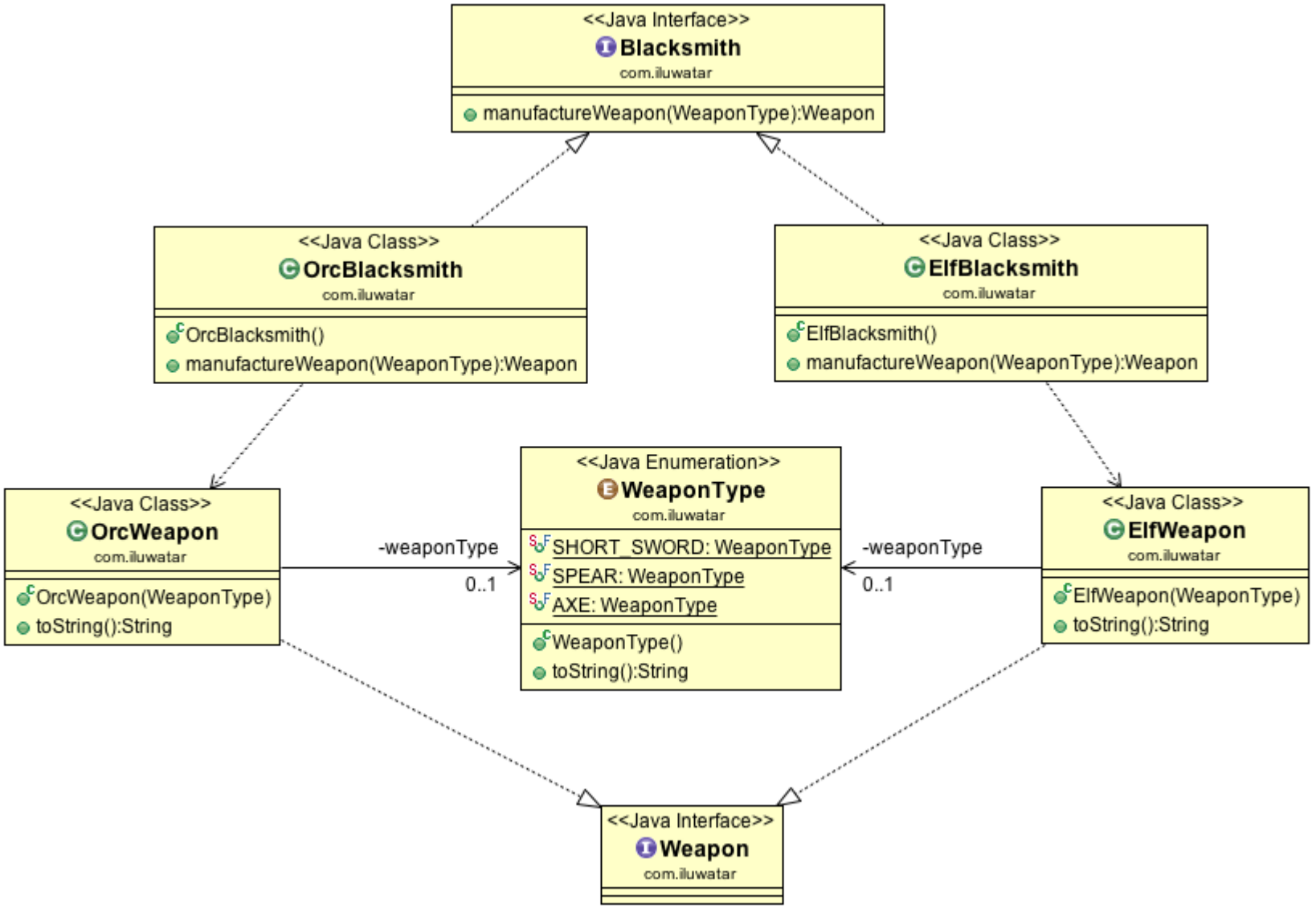
Factory method (1)

- **Aim**:
  - It provides an interface to create an object but it lets the subclasses to decide which class is instantiated

Factory method (2)

- **Example**: https://github.com/iluwatar/java-design-patterns/tree/master/factory-method

Factory method (3)



Factory method (4)

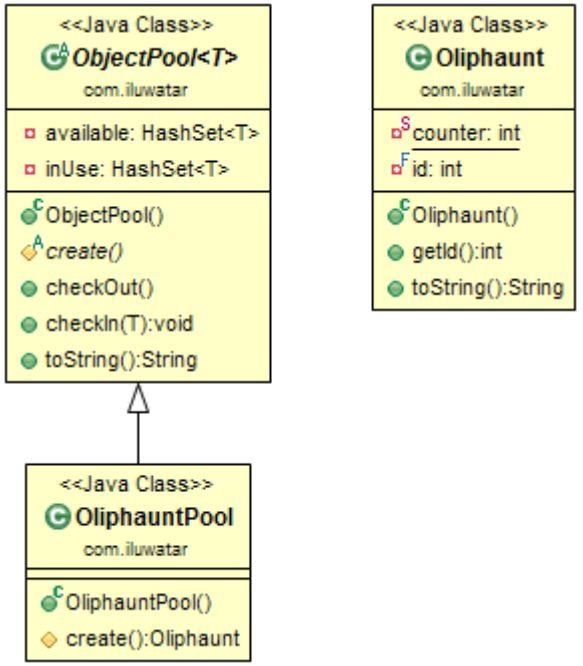
- **Known uses**:
  - `javax.xml.parsers.DocumentBuilderFactory#newDocumentBuilder()`
    https://docs.oracle.com/javase/8/docs/api/javax/xml/parsers/DocumentBuilderFactory.html#newDocumentBuilder--
  - `javax.xml.parsers.SAXParserFactory#newSAXParser()`
    https://docs.oracle.com/javase/8/docs/api/javax/xml/parsers/SAXParserFactory.html#newSAXParser--

Object pool (1)

- **Aim**: Registers a set of initialized objects for the service of the demands instead of erasing and re-instantiating them

Object pool (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/object-pool



Object pool (3)

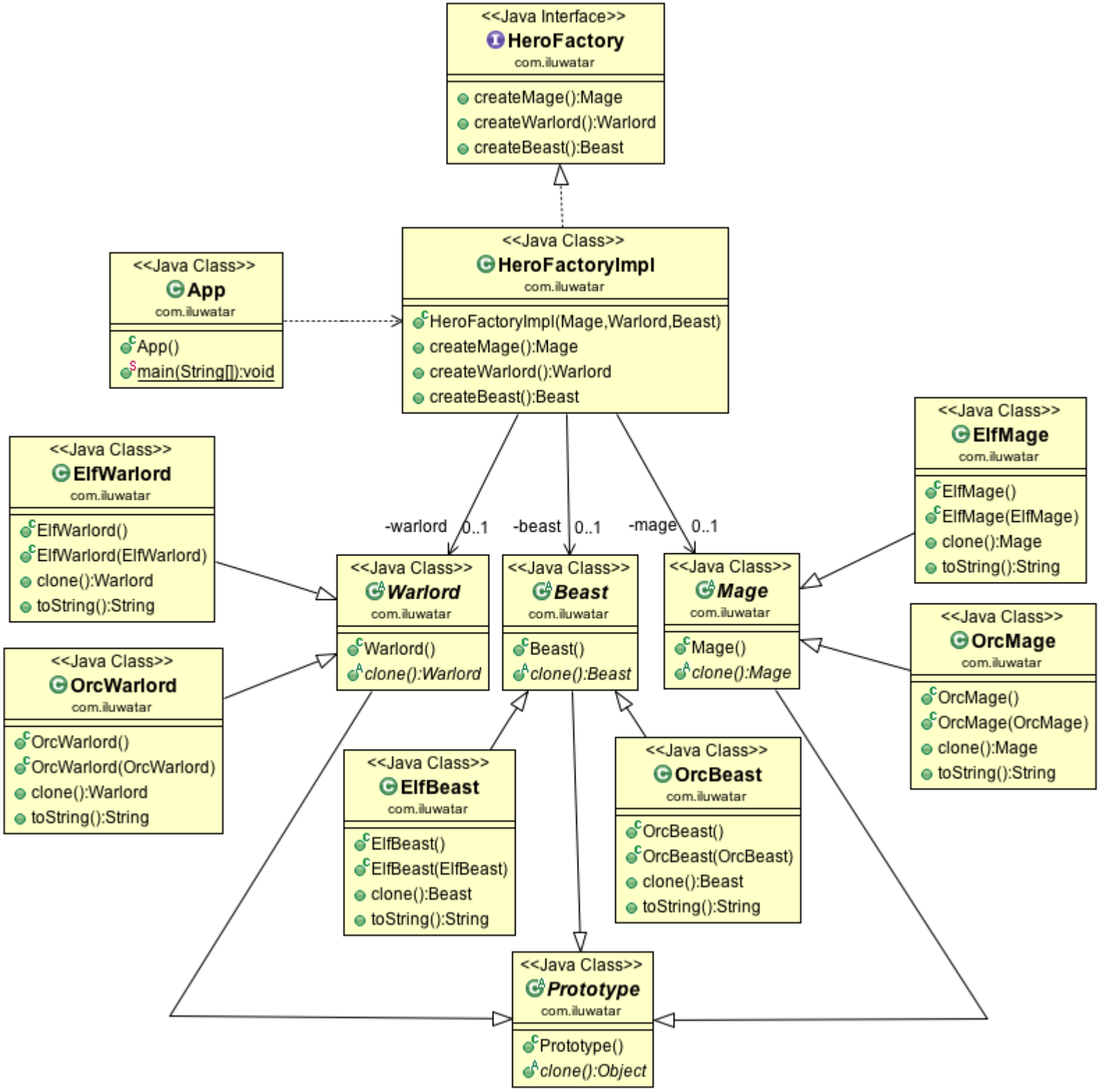
- **Known uses**:
  - Pooling database connections (*connection pooling*)
    - E.g.:
      - *Apache Commons DBCP* https://commons.apache.org/proper/commons-dbcp/
      - *HikariCP* https://brettwooldridge.github.io/HikariCP/
      - *Vibur DBCP* http://www.vibur.org/

Prototype (1)

- **Aim**: By providing a prototype instance it defines what type of objects are to be created. The object are copies of this prototype

Prototype (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/prototype


<<Java Interface>>
HeroFactory
com.iluwatar
createMage():Mage
createWarlord():Warlord
createBeast():Beast
<<Java Class>>
HeroFactoryImpl
com.iluwatar
HeroFactoryImpl(Mage,Warlord,Beast)
createMage():Mage
createWarlord():Warlord
createBeast():Beast
<<Java Class>>
App
com.iluwatar
App()
main(String[]):void
-warlord 0..1
-beast 0..1
-mage 0..1
<<Java Class>>
ElfWarlord
com.iluwatar
ElfWarlord()
ElfWarlord(ElfWarlord)
clone():Warlord
toString():String
<<Java Class>>
OrcWarlord
com.iluwatar
OrcWarlord()
OrcWarlord(OrcWarlord)
clone():Warlord
toString():String
<<Java Class>>
Warlord
com.iluwatar
Warlord()
clone():Warlord
<<Java Class>>
Beast
com.iluwatar
Beast()
clone():Beast
<<Java Class>>
Mage
com.iluwatar
Mage()
clone():Mage
<<Java Class>>
ElfMage
com.iluwatar
ElfMage()
ElfMage(ElfMage)
clone():Mage
toString():String
<<Java Class>>
OrcMage
com.iluwatar
OrcMage()
OrcMage(OrcMage)
clone():Mage
toString():String
<<Java Class>>
ElfBeast
com.iluwatar
ElfBeast()
ElfBeast(ElfBeast)
clone():Beast
toString():String
<<Java Class>>
OrcBeast
com.iluwatar
OrcBeast()
OrcBeast(OrcBeast)
clone():Beast
toString():String
<<Java Class>>
Prototype
com.iluwatar
Prototype()
clone():Object

Prototype (4)

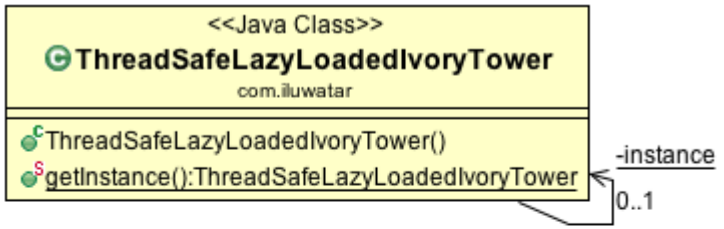
- **Known uses**:
  - `java.lang.Cloneable` https://docs.oracle.com/javase/8/docs/api/java/lang/Cloneable.html

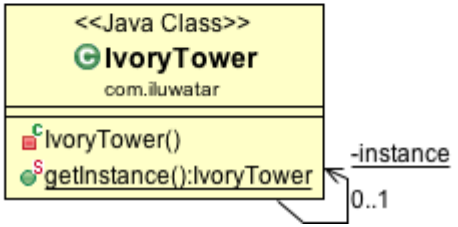
Singleton (1)

Aim: It lets only one instance to be created from a class. For this it provides a global acces point.

Singleton (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/singleton





Singleton (3)

- **Known uses**:
  - `java.lang.Runtime` https://docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html
  - `java.time.chrono.IsoChronology` https://docs.oracle.com/javase/8/docs/api/java/time/chrono/IsoChronology.html

More creational patterns

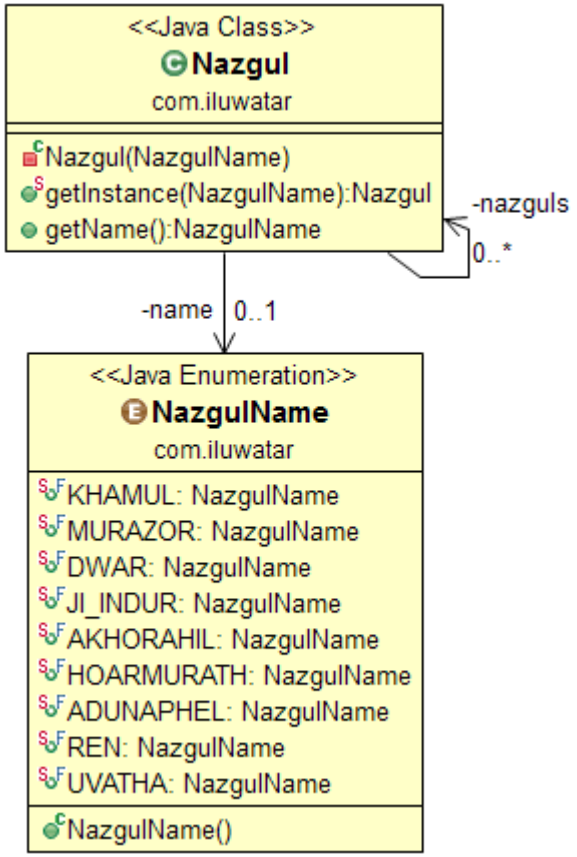
- *Factory Kit* https://github.com/iluwatar/java-design-patterns/tree/master/factory-kit
- *Module* https://github.com/iluwatar/java-design-patterns/tree/master/module
- *Monostate* https://github.com/iluwatar/java-design-patterns/tree/master/monostate
- *Multiton* https://github.com/iluwatar/java-design-patterns/tree/master/multiton
- *Object Mother* https://github.com/iluwatar/java-design-patterns/tree/master/object-mother
- *Step Builder* https://github.com/iluwatar/java-design-patterns/tree/master/step-builder
- *Value Object* https://github.com/iluwatar/java-design-patterns/tree/master/value-object
- …

Multition (1)

- **Aim**: It lets only a given number of instances to be created from a class. For them it provides a global acces point.

Multition (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/multiton



Multition (3)

- **Known uses**:
  - `java.lang.Thread.State` https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html
  - `java.math.RoundingMode` http://docs.oracle.com/javase/8/docs/api/java/math/RoundingMode.html
  - `java.time.DayOfWeek` https://docs.oracle.com/javase/8/docs/api/java/time/DayOfWeek.html
  - `java.time.chrono.IsoEra` http://docs.oracle.com/javase/8/docs/api/java/time/chrono/IsoEra.html

Structural patterns (GoF)

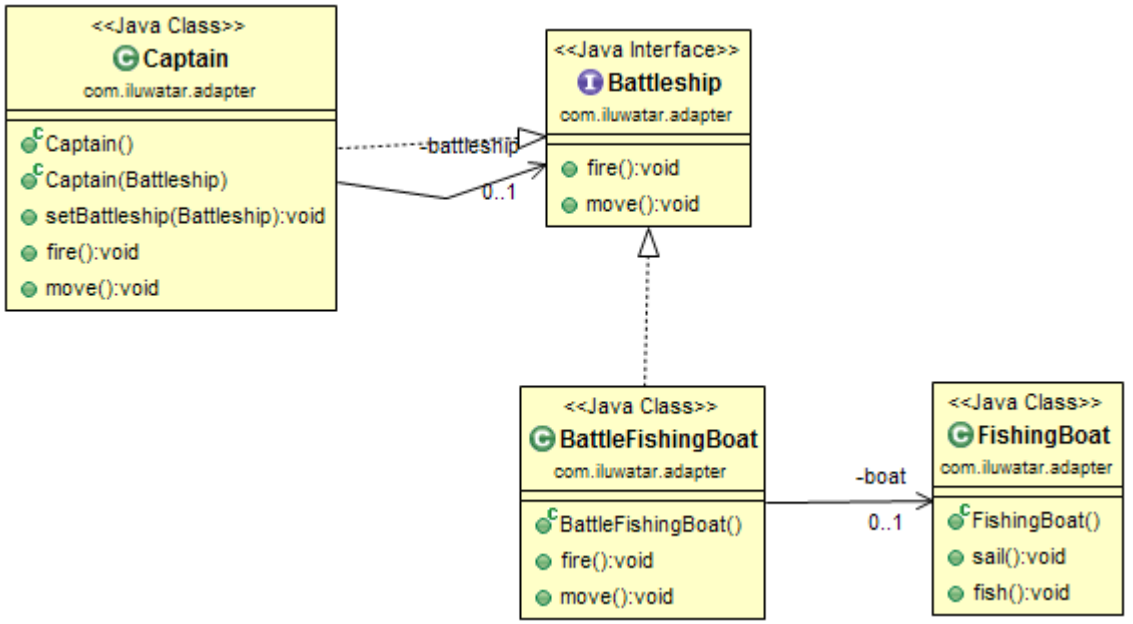
- *Adapter*
- *Bridge*
- *Composite*
- *Decorator*
- *Facade*
- *Flyweight*
- *Proxy*

Adapter (1)

- **Aim**:
  - It converts the interface of a class to the interface that can be used by other classes.
  - By the use of it classes with incompatible intefaces may work together

Adapter (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/adapter



Adapter (3)

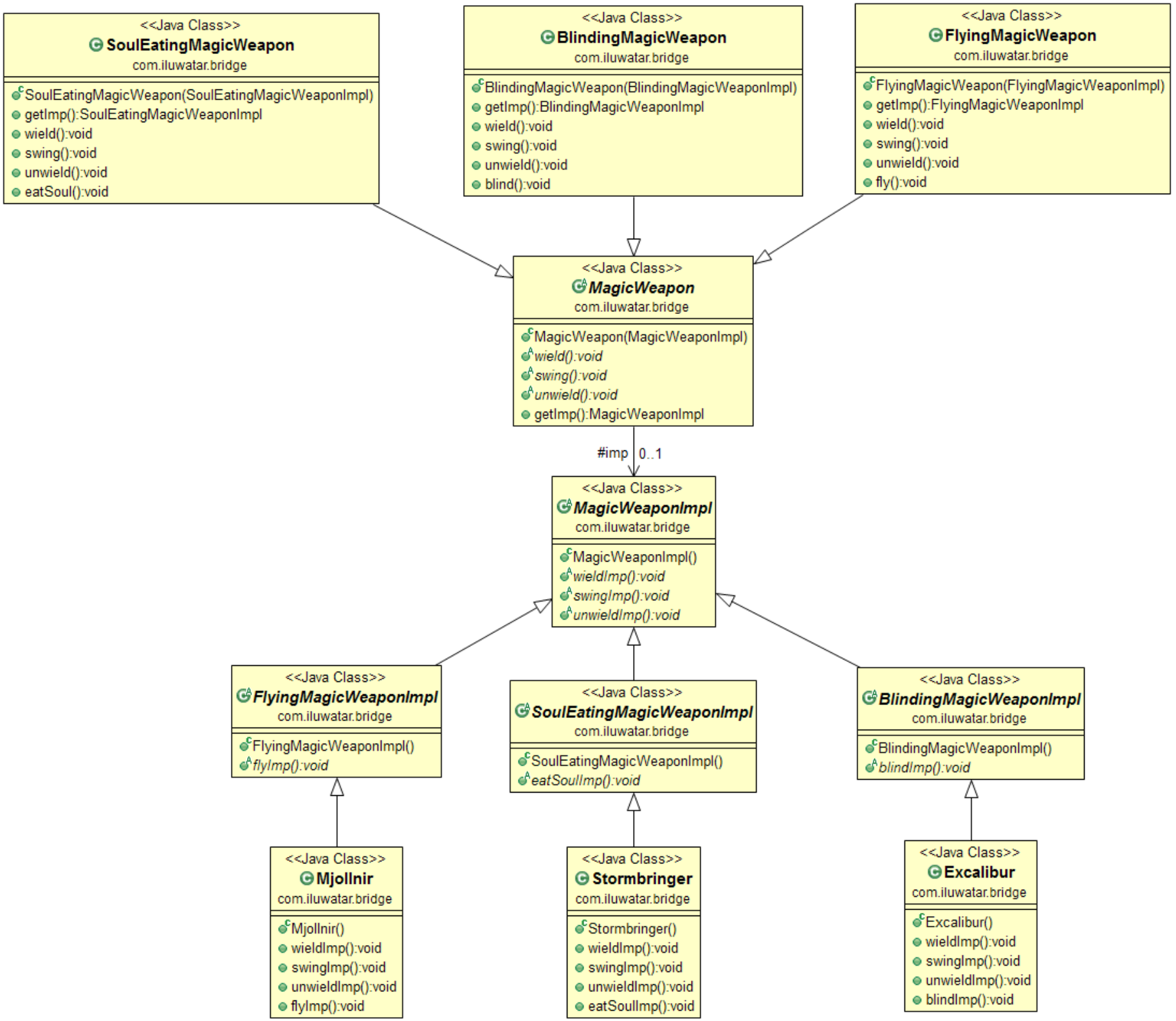
- **Known uses**:
  - `java.io.InputStreamReader`
    http://docs.oracle.com/javase/8/docs/api/java/io/InputStreamReader.html
  - `java.io.OutputStreamWriter`
    http://docs.oracle.com/javase/8/docs/api/java/io/OutputStreamWriter.html
  - `javax.xml.bind.annotation.adapters.XmlAdapter`
    https://docs.oracle.com/javase/8/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html
  - `org.xml.sax.helpers.ParserAdapter`
    https://docs.oracle.com/javase/8/docs/api/org/xml/sax/helpers/ParserAdapter.html
  - `org.xml.sax.helpers.XMLReaderAdapter`
    https://docs.oracle.com/javase/8/docs/api/org/xml/sax/helpers/XMLReaderAdapter.html

Bridge (1)

Aim: It separates the abstract representation from the implementation so that they can be modified separately.

Bridge (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/bridge


<<Java Class>>
SoulEatingMagicWeapon
com.iluwatar.bridge
SoulEatingMagicWeapon(SoulEatingMagicWeaponImpl)
getImp():SoulEatingMagicWeaponImpl
wield():void
swing():void
unwield():void
eatSoul():void
<<Java Class>>
BlindingMagicWeapon
com.iluwatar.bridge
BlindingMagicWeapon(BlindingMagicWeaponImpl)
getImp():BlindingMagicWeaponImpl
wield():void
swing():void
unwield():void
blind():void
<<Java Class>>
FlyingMagicWeapon
com.iluwatar.bridge
FlyingMagicWeapon(FlyingMagicWeaponImpl)
getImp():FlyingMagicWeaponImpl
wield():void
swing():void
unwield():void
fly():void
<<Java Class>>
MagicWeapon
com.iluwatar.bridge
MagicWeapon(MagicWeaponImpl)
wield():void
swing():void
unwield():void
getImp():MagicWeaponImpl
#imp
0..1
<<Java Class>>
MagicWeaponImpl
com.iluwatar.bridge
MagicWeaponImpl()
wieldImp():void
swingImp():void
unwieldImp():void
<<Java Class>>
FlyingMagicWeaponImpl
com.iluwatar.bridge
FlyingMagicWeaponImpl()
flyImp():void
<<Java Class>>
SoulEatingMagicWeaponImpl
com.iluwatar.bridge
SoulEatingMagicWeaponImpl()
eatSoulImp():void
<<Java Class>>
BlindingMagicWeaponImpl
com.iluwatar.bridge
BlindingMagicWeaponImpl()
blindImp():void
<<Java Class>>
Mjollnir
com.iluwatar.bridge
Mjollnir()
wieldImp():void
swingImp():void
unwieldImp():void
flyImp():void
<<Java Class>>
Stormbringer
com.iluwatar.bridge
Stormbringer()
wieldImp():void
swingImp():void
unwieldImp():void
eatSoulImp():void
<<Java Class>>
Excalibur
com.iluwatar.bridge
Excalibur()
wieldImp():void
swingImp():void
unwieldImp():void
blindImp():void

Bridge (4)

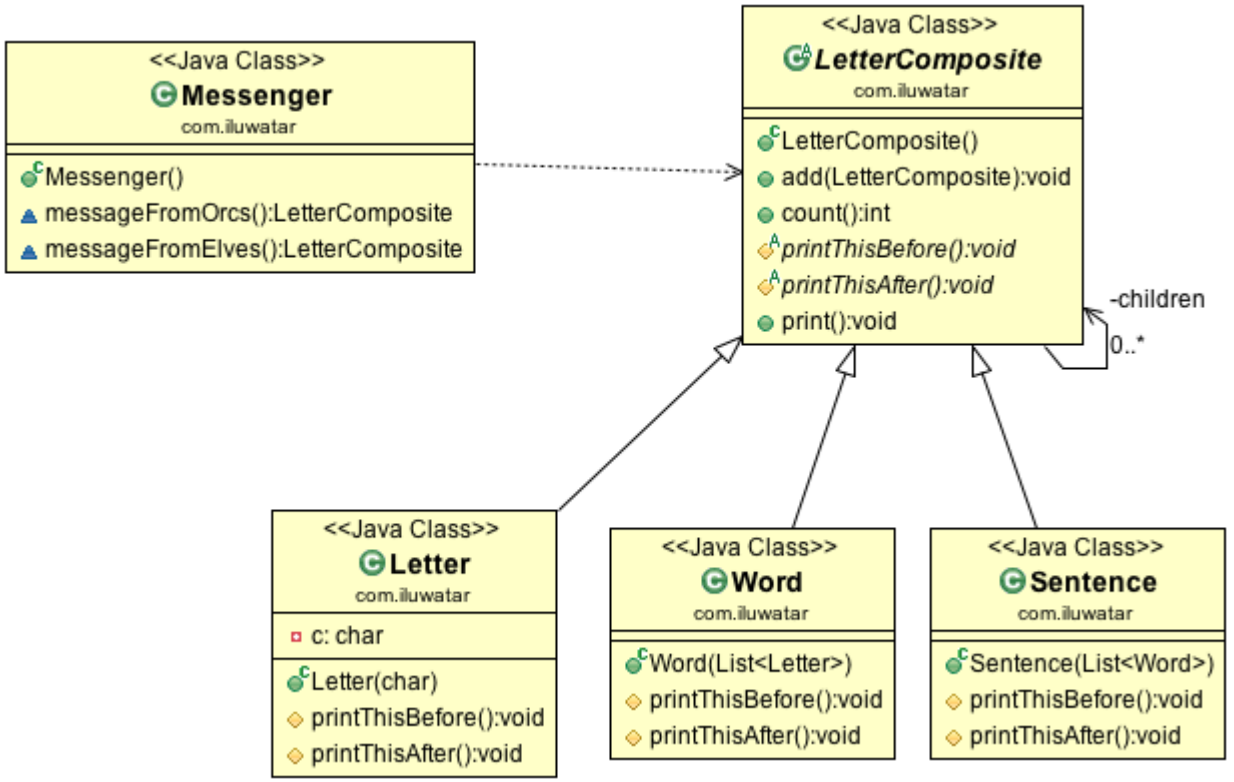
- **Known uses**:
  - *JDBC* https://jcp.org/en/jsr/detail?id=221

Composite (1)

- **Aim**:
  - The objects are structured as a tree in order to illustrate the part-whole relations
  - As a result the separate objects and the composits may be handled as well

Composite (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/composite



Composite (3)

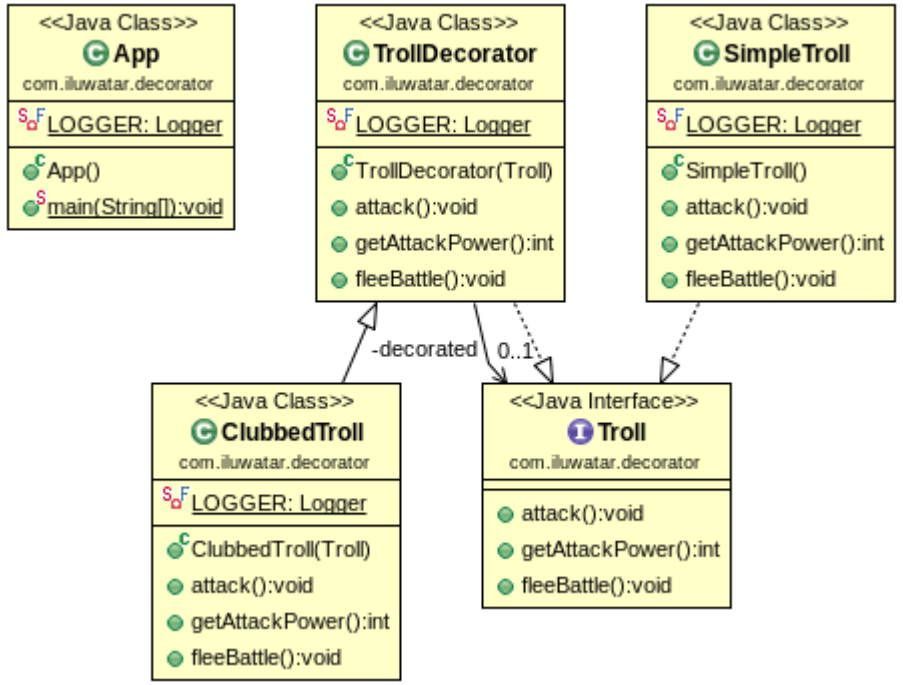
- **Known uses:**
  - `javafx.scene.Node` https://docs.oracle.com/javafx/2/api/javafx/scene/Node.html
  - *jsoup: Java HTML Parser* https://jsoup.org/
    - `org.jsoup.nodes.Node` https://jsoup.org/apidocs/org/jsoup/nodes/Node.html

Decorator (1)

- **Aim:**
  - Adds more additional responsibilities to classes dinamically.
  - Can be used as an alternative of sub-classes.

Decorator (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/decorator



Decorator (3)

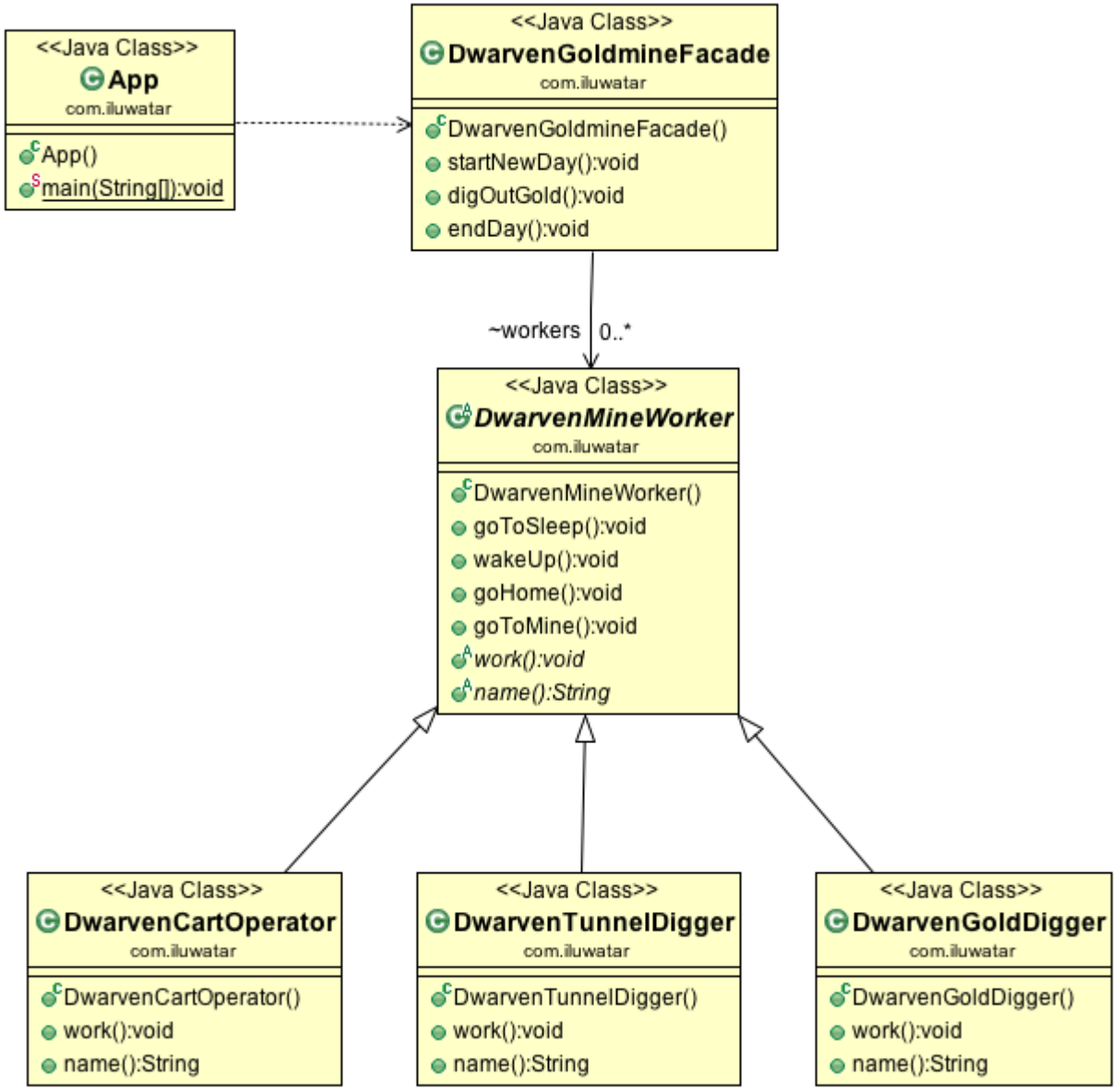
- **Known uses**:
  - `java.io.InputStream`
    http://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html
    - Where the constructor gets `InputStream` objects, like `java.io.ObjectInputStream`.
  - `java.io.OutputStream`
    http://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html
    - Where the constructor gets `OutputStream` objects, like `java.io.ObjectOutputStream`.
  - `java.util.Collections#unmodifiable`*`XXX`*`()`
    https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html

Facade (1)

- **Aim**:
  - In a sub-system it provides a main interface that gathers a set of other interfaces.
  - By the use of it a higher level interface is provided in order to make the use of the subsystem more easy.

Facade (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/facade


<<Java Class>>
App
com.iluwatar
App()
main(String[]):void
<<Java Class>>
DwarvenGoldmineFacade
com.iluwatar
DwarvenGoldmineFacade()
startNewDay():void
digOutGold():void
endDay():void
~workers
0..*
<<Java Class>>
DwarvenMineWorker
com.iluwatar
DwarvenMineWorker()
goToSleep():void
wakeUp():void
goHome():void
goToMine():void
work():void
name():String
<<Java Class>>
DwarvenCartOperator
com.iluwatar
DwarvenCartOperator()
work():void
name():String
<<Java Class>>
DwarvenTunnelDigger
com.iluwatar
DwarvenTunnelDigger()
work():void
name():String
<<Java Class>>
DwarvenGoldDigger
com.iluwatar
DwarvenGoldDigger()
work():void
name():String

Facade (4)

- **Known uses**:
  - *Simple Logging Facade for Java (SLF4J)*
    https://www.slf4j.org/

Flyweight (1)

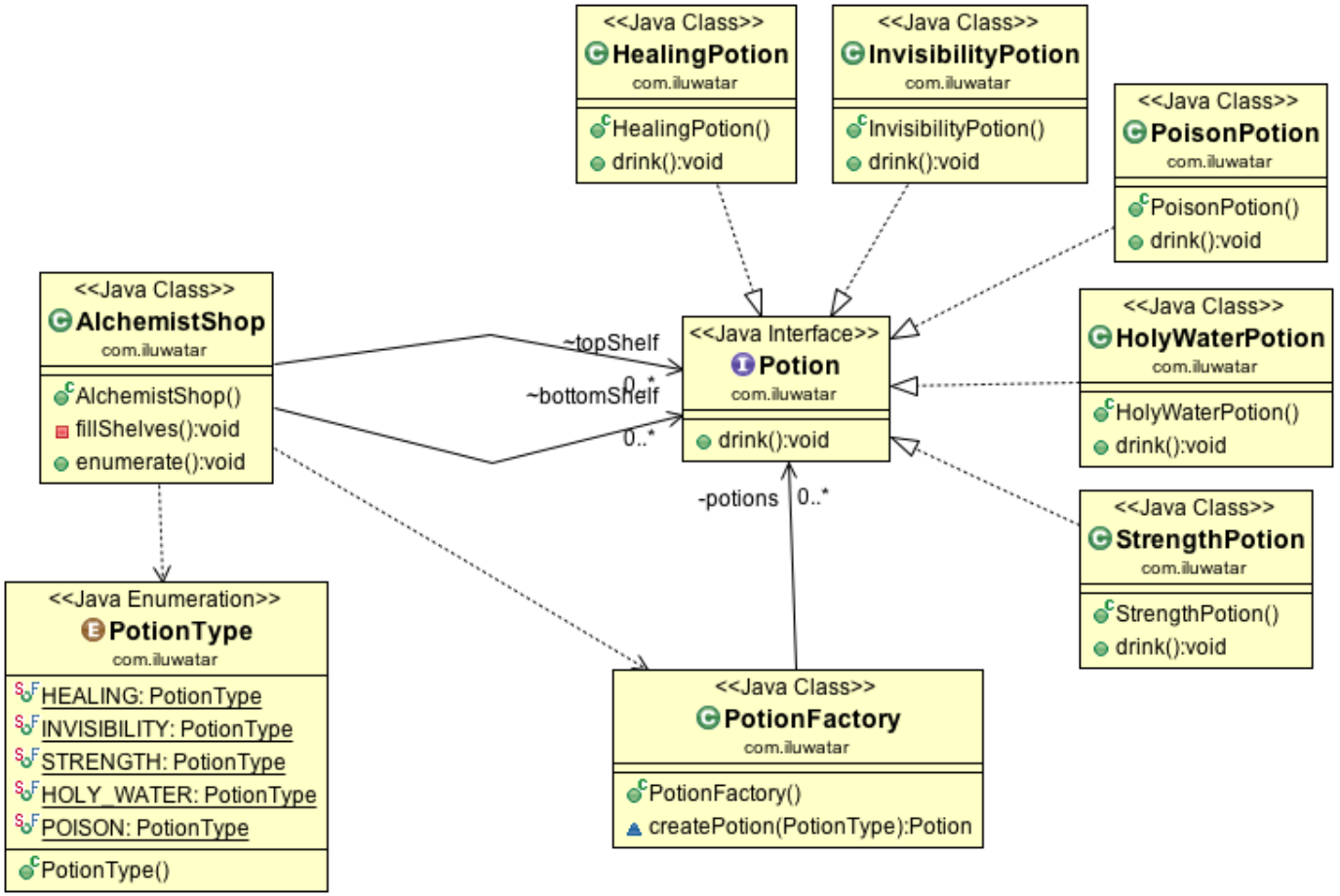
Aim:

- Supports the use of high mass granular objects.

Flyweight (2)

- **Example**: https://github.com/iluwatar/java-design-patterns/tree/master/flyweight

Flyweight (3)


<<Java Class>>
HealingPotion
com.iluwatar
HealingPotion()
drink():void
<<Java Class>>
InvisibilityPotion
com.iluwatar
InvisibilityPotion()
drink():void
<<Java Class>>
PoisonPotion
com.iluwatar
PoisonPotion()
drink():void
<<Java Class>>
AlchemistShop
com.iluwatar
AlchemistShop()
fillShelves():void
enumerate():void
~topShelf
0..*
~bottomShelf
0..*
<<Java Interface>>
Potion
com.iluwatar
drink():void
<<Java Class>>
HolyWaterPotion
com.iluwatar
HolyWaterPotion()
drink():void
-potions
0..*
<<Java Class>>
StrengthPotion
com.iluwatar
StrengthPotion()
drink():void
<<Java Enumeration>>
PotionType
com.iluwatar
HEALING: PotionType
INVISIBILITY: PotionType
STRENGTH: PotionType
HOLY_WATER: PotionType
POISON: PotionType
PotionType()
<<Java Class>>
PotionFactory
com.iluwatar
PotionFactory()
createPotion(PotionType):Potion

Flyweight (4)

- **Known uses**:
  - `java.lang.Byte#valueOf(byte b)` https://docs.oracle.com/javase/8/docs/api/java/lang/Byte.html#valueOf-byte-
  - `java.lang.Character#valueOf(char c)` https://docs.oracle.com/javase/8/docs/api/java/lang/Character.html#valueOf-char-
  - `java.lang.Integer#valueOf(int i)` https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html#valueOf-int-

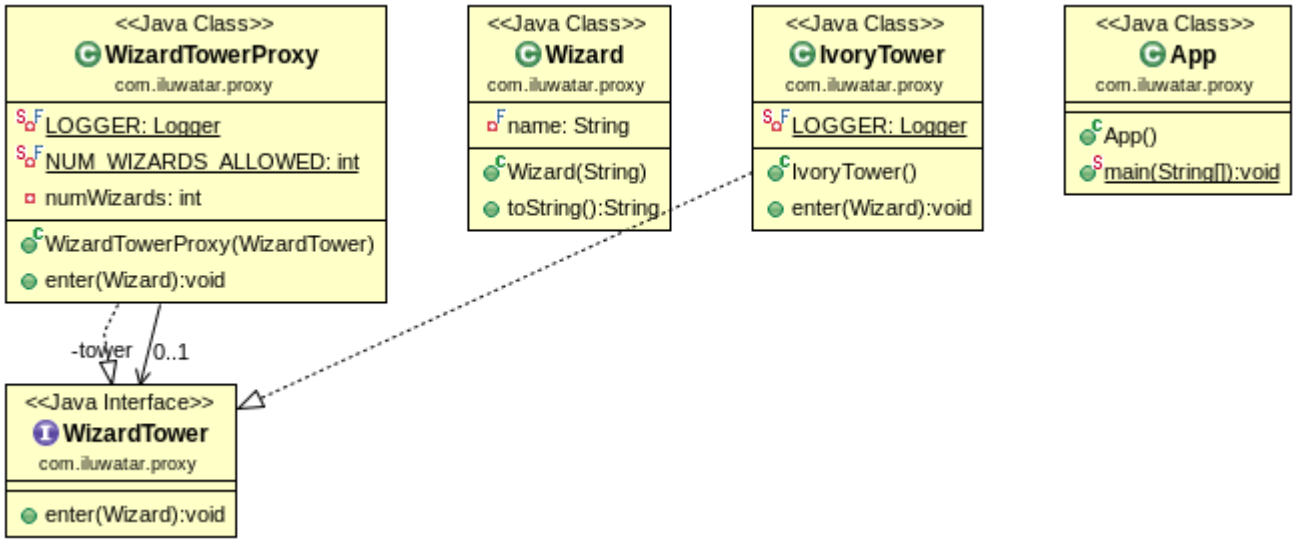
Proxy (1)

Aim:

- A given object is replaced by another one that overrides the access to the original object as well.

Proxy (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/proxy



Proxy (3)

- **Known uses**:
  - `java.lang.reflect.Proxy` http://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Proxy.html
  - *Apache Commons Proxy* https://commons.apache.org/proper/commons-proxy/

More structural patterns

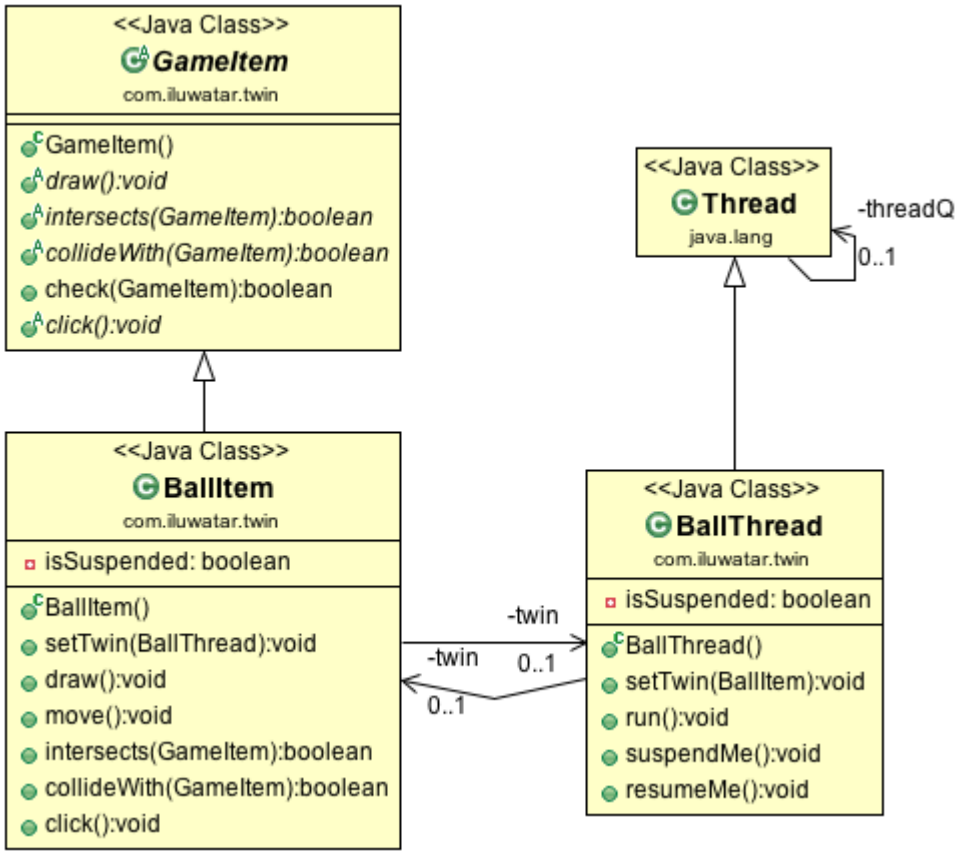
- *Abstract Document* https://github.com/iluwatar/java-design-patterns/tree/master/abstract-document
- *Module* https://github.com/iluwatar/java-design-patterns/tree/master/module
- *Private Class Data* https://github.com/iluwatar/java-design-patterns/tree/master/private-class-data
- *Twin* https://github.com/iluwatar/java-design-patterns/tree/master/twin
- …

Twin (1)

- **Aim:**
  - Model multiple inheritance in languages where it is not supported by default.

Twin (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/twin



Behavioral petterns (GoF)

- *Chain of Responsibility*
- *Command*
- *Interpreter*
- *Iterator*
- *Mediator*
- *Memento*
- *Observer*
- *State*
- *Strategy*
- *Template Method*
- *Visitor*

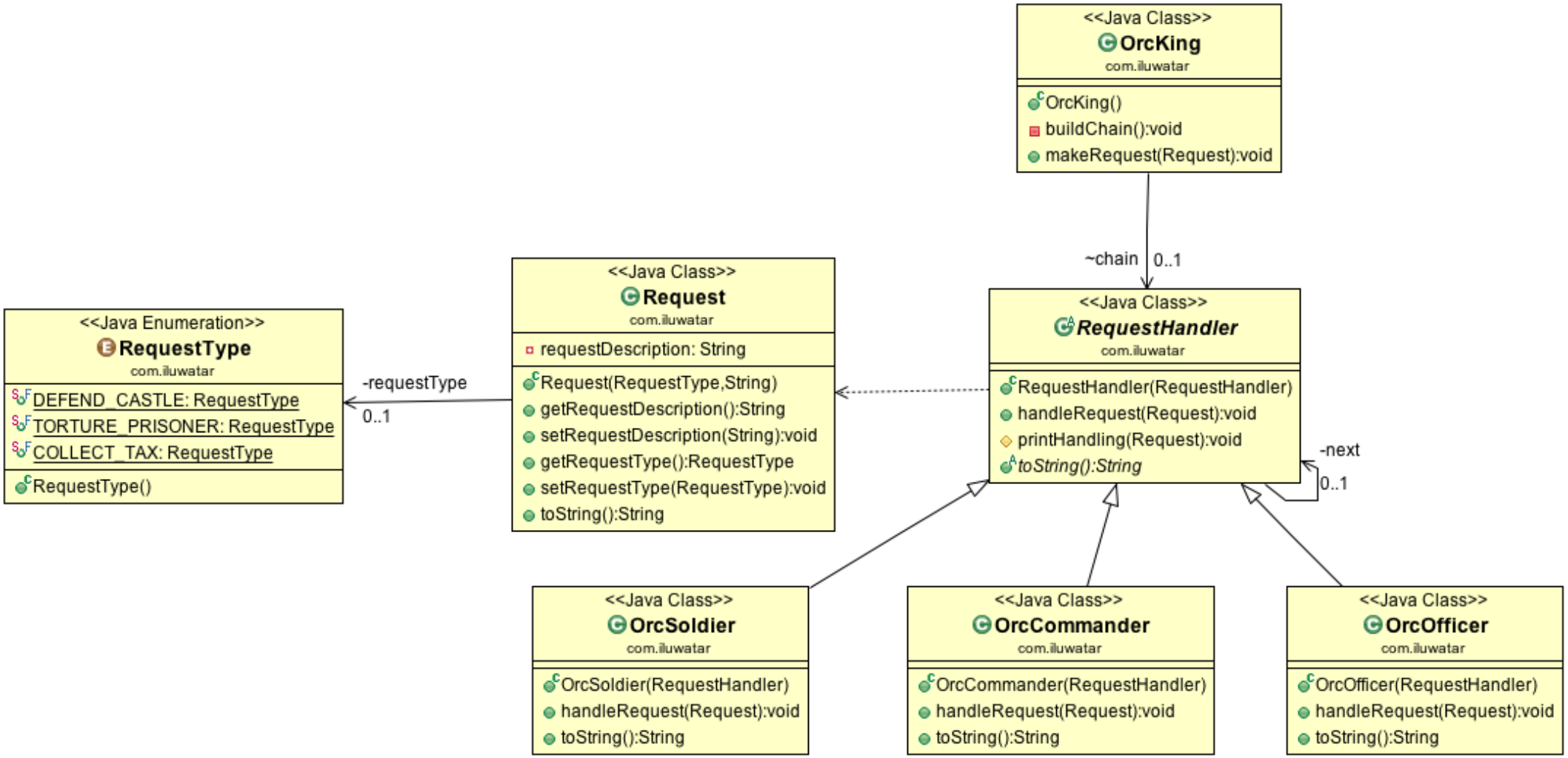
Chain of Responsibility (1)

- **Aim**:
  - The aim is to avoid the connection of the sender of the request to the receiver
  - We do this by providing the right to handle the request for more objects.
  - The receiver objects may be chained. The object goes in the chain while it reaches such an object that can handle it.

Chain of Responsibility (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/chain

Chain of Responsibility (3)



Chain of Responsibility (4)

- **Known uses**:
  - `java.util.logging.Logger`
    https://docs.oracle.com/javase/8/docs/api/java/util/logging/Logger.html

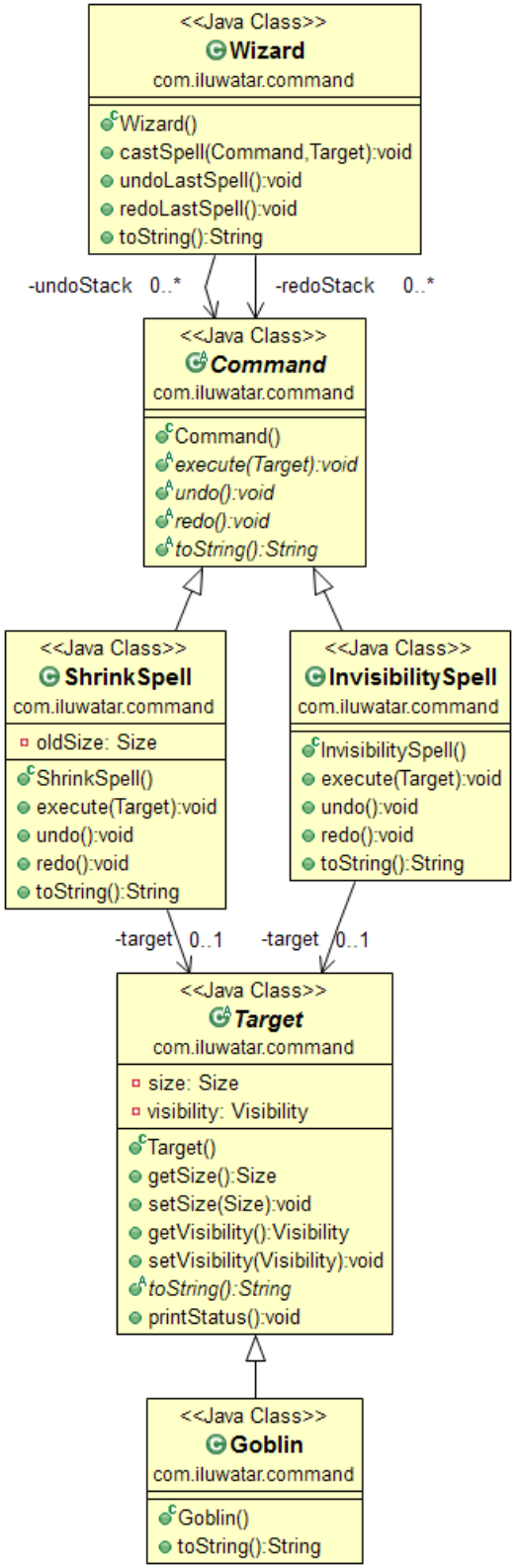
Command (1)

Aim:

- It closes queries to an object in order to make it possible to give these queries as parameters to other agents

Command (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/command


<<Java Class>>
Wizard
com.iluwatar.command
Wizard()
castSpell(Command,Target):void
undoLastSpell():void
redoLastSpell():void
toString():String
-undoStack 0..*
-redoStack 0..*
<<Java Class>>
Command
com.iluwatar.command
Command()
execute(Target):void
undo():void
redo():void
toString():String
<<Java Class>>
ShrinkSpell
com.iluwatar.command
oldSize: Size
ShrinkSpell()
execute(Target):void
undo():void
redo():void
toString():String
<<Java Class>>
InvisibilitySpell
com.iluwatar.command
InvisibilitySpell()
execute(Target):void
undo():void
redo():void
toString():String
-target 0..1
-target 0..1
<<Java Class>>
Target
com.iluwatar.command
size: Size
visibility: Visibility
Target()
getSize():Size
setSize(Size):void
getVisibility():Visibility
setVisibility(Visibility):void
toString():String
printStatus():void
<<Java Class>>
Goblin
com.iluwatar.command
Goblin()
toString():String

Command (4)

- **Known uses**:
  - `java.lang.Runnable` https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html

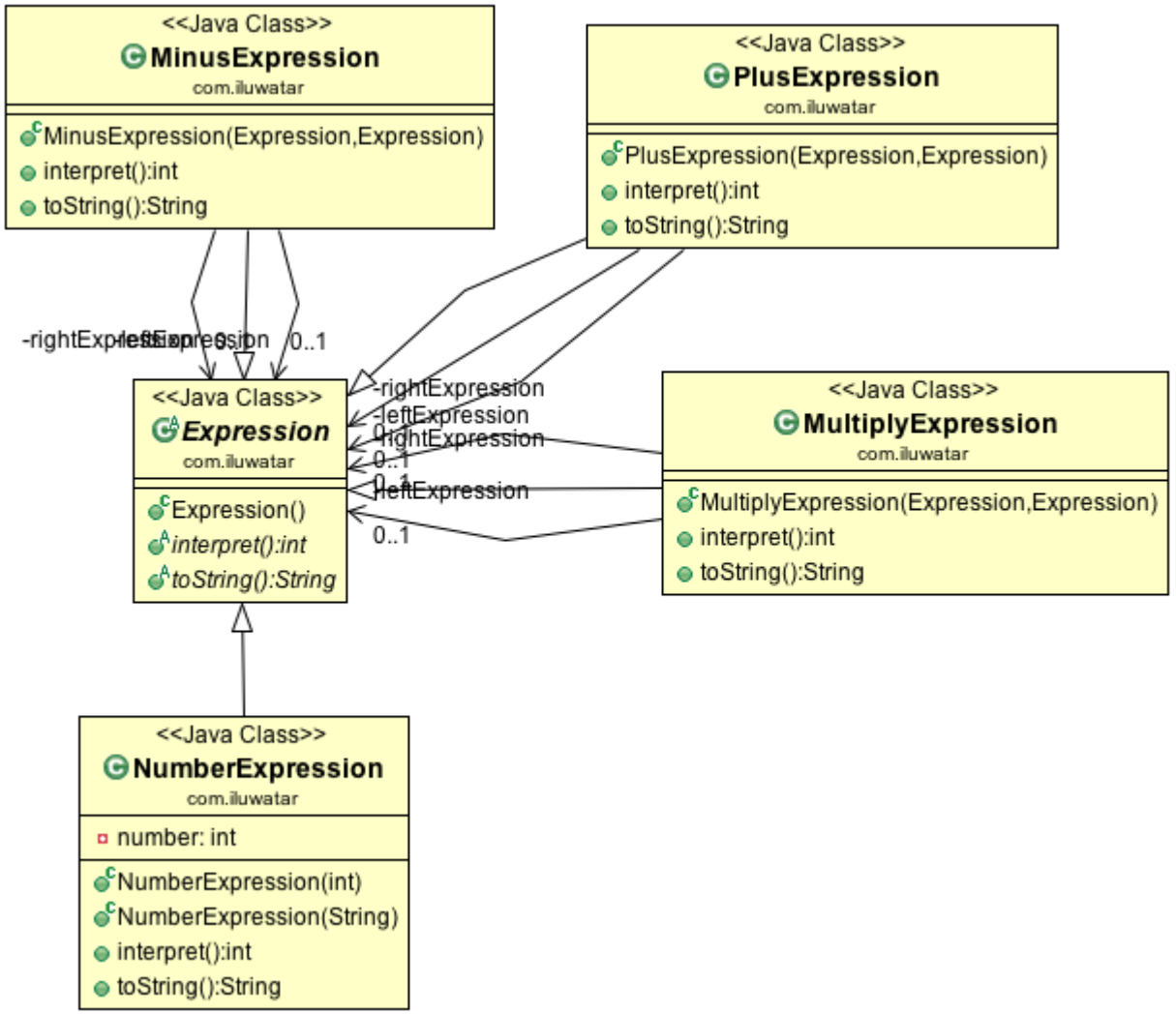
Interpreter (1)

- **Aim**: Represents the grammer of a given language and provide an interpreter for it that understands the sentences of that language.

Interpreter (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/interpreter

Interpreter (3)



Interpreter (4)

- **Known uses**:
  - `java.text.Format`
    https://docs.oracle.com/javase/8/docs/api/java/text/Format.html
  - `java.time.format.DateTimeFormatter`
    https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html
  - `java.util.regex.Pattern`
    https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html

Iterator (1)

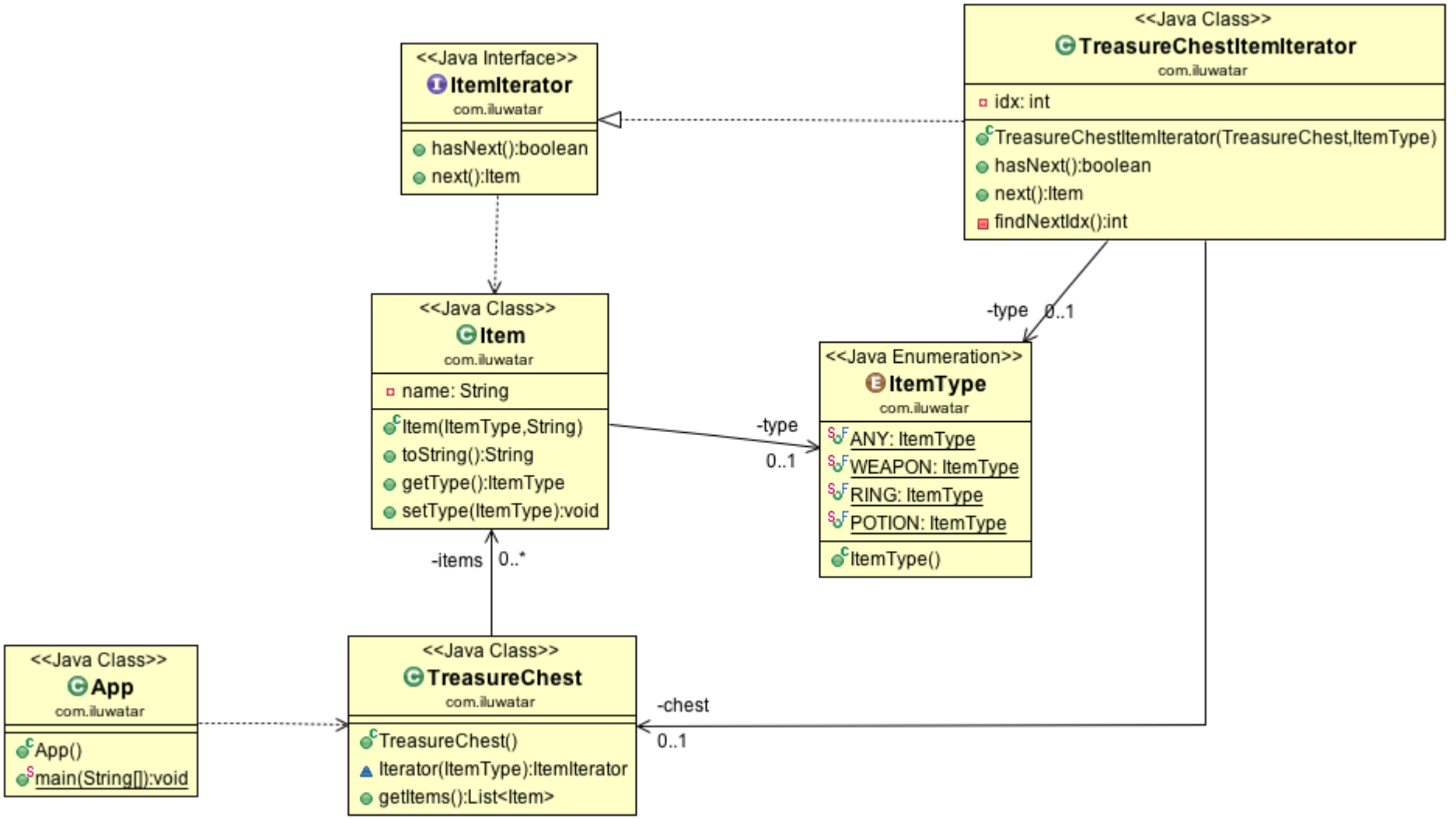
Aim:

- It provides a sequential access to elements of complex objects without showing the representation of the objects

Iterator (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/iterator

Iterator (3)


<<Java Interface>>
ItemIterator
com.iluwatar
hasNext():boolean
next():Item
<<Java Class>>
TreasureChestItemIterator
com.iluwatar
idx: int
TreasureChestItemIterator(TreasureChest,ItemType)
hasNext():boolean
next():Item
findNextIdx():int
-type
0..1
<<Java Class>>
Item
com.iluwatar
name: String
Item(ItemType,String)
toString():String
getType():ItemType
setType(ItemType):void
-type
0..1
<<Java Enumeration>>
ItemType
com.iluwatar
ANY: ItemType
WEAPON: ItemType
RING: ItemType
POTION: ItemType
ItemType()
-items
0..*
<<Java Class>>
App
com.iluwatar
App()
main(String[]):void
<<Java Class>>
TreasureChest
com.iluwatar
TreasureChest()
Iterator(ItemType):ItemIterator
getItems():List<Item>
-chest
0..1

Iterator (4)

- **Known uses**:
  - `java.util.Enumeration` https://docs.oracle.com/javase/8/docs/api/java/util/Enumeration.html
  - `java.util.Iterator` https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

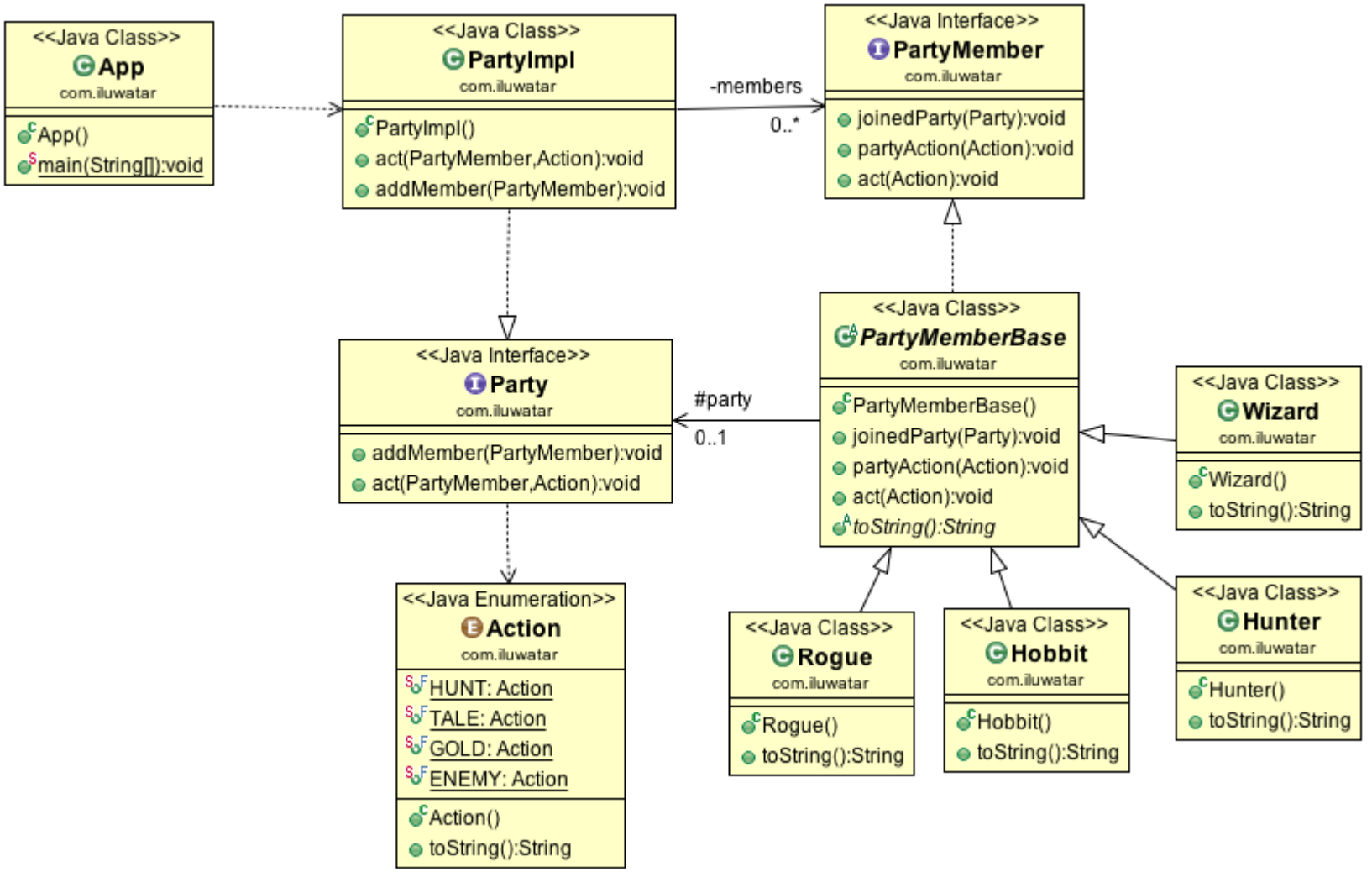
Mediator (1)

- **Aim**:
  - The aim is to provide an object that controls the the cooperation of a set of objects
  - By this mmethod we create a weak connection where the objects do not refer to each other and the connectins between them may be modified independently of each other

Mediator (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/mediator

Mediator (3)


<<Java Class>>
App
com.iluwatar
App()
main(String[]):void
<<Java Class>>
PartyImpl
com.iluwatar
PartyImpl()
act(PartyMember,Action):void
addMember(PartyMember):void
-members
0..*
<<Java Interface>>
PartyMember
com.iluwatar
joinedParty(Party):void
partyAction(Action):void
act(Action):void
<<Java Interface>>
Party
com.iluwatar
addMember(PartyMember):void
act(PartyMember,Action):void
#party
0..1
<<Java Class>>
PartyMemberBase
com.iluwatar
PartyMemberBase()
joinedParty(Party):void
partyAction(Action):void
act(Action):void
toString():String
<<Java Class>>
Wizard
com.iluwatar
Wizard()
toString():String
<<Java Enumeration>>
Action
com.iluwatar
HUNT: Action
TALE: Action
GOLD: Action
ENEMY: Action
Action()
toString():String
<<Java Class>>
Rogue
com.iluwatar
Rogue()
toString():String
<<Java Class>>
Hobbit
com.iluwatar
Hobbit()
toString():String
<<Java Class>>
Hunter
com.iluwatar
Hunter()
toString():String

Mediator (4)

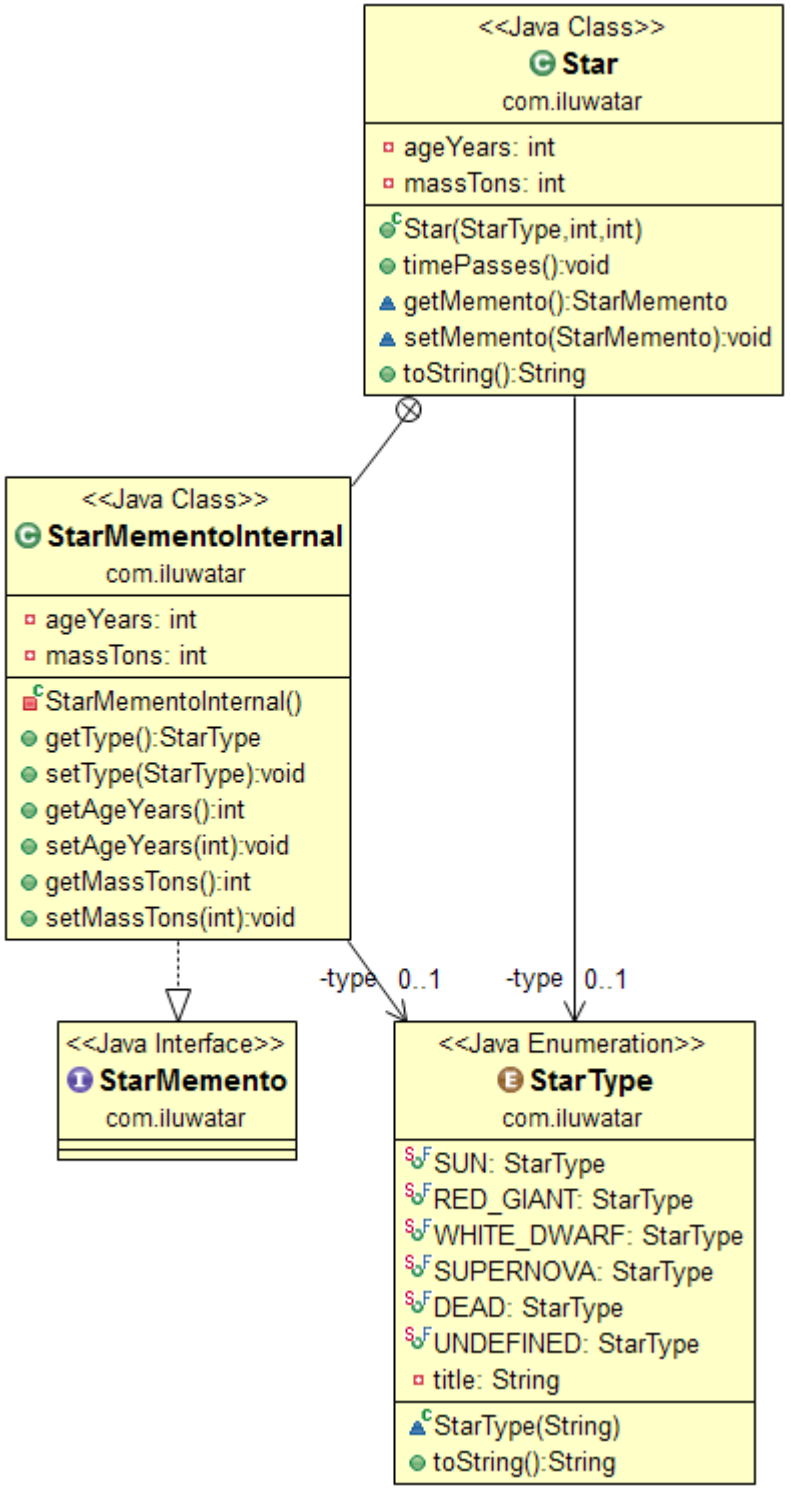
- **Known uses**:
  - `java.util.Timer`
    https://docs.oracle.com/javase/8/docs/api/java/util/Timer.html

Memento (1)

- **Aim**:
  - Save the show the inner state of an object without breaking enclosure so that later the object can be loaded back to this previous state

Memento (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/memento


<<Java Class>>
Star
com.iluwatar
ageYears: int
massTons: int
Star(StarType,int,int)
timePasses():void
getMemento():StarMemento
setMemento(StarMemento):void
toString():String
<<Java Class>>
StarMementoInternal
com.iluwatar
ageYears: int
massTons: int
StarMementoInternal()
getType():StarType
setType(StarType):void
getAgeYears():int
setAgeYears(int):void
getMassTons():int
setMassTons(int):void
-type 0..1
-type 0..1
<<Java Interface>>
StarMemento
com.iluwatar
<<Java Enumeration>>
StarType
com.iluwatar
SUN: StarType
RED_GIANT: StarType
WHITE_DWARF: StarType
SUPERNOVA: StarType
DEAD: StarType
UNDEFINED: StarType
title: String
StarType(String)
toString():String

Memento (4)

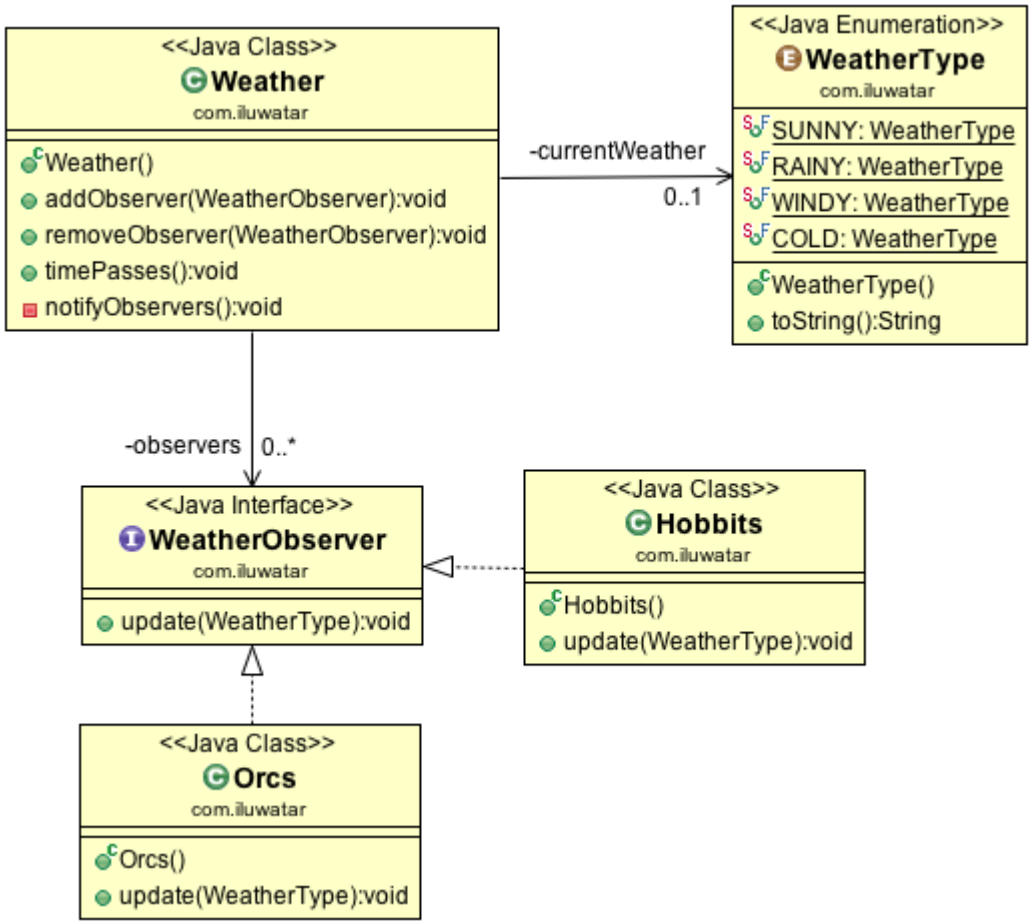
- **Known uses**:
  - `java.util.Date` https://docs.oracle.com/javase/8/docs/api/java/util/Date.html
    - See `getTime()` and `setTime(long time)` methods.

Observer (1)

- **Aim**:
  - Provides a connection between objects so that if an object changes all other objects depending on it are informed about that

Observer (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/observer



Observer (3)

- **Known uses**:
  - `java.util.EventListener` https://docs.oracle.com/javase/8/docs/api/java/util/EventListener.html
  - `java.util.Observer` https://docs.oracle.com/javase/8/docs/api/java/util/Observer.html
    `java.util.Observable` https://docs.oracle.com/javase/8/docs/api/java/util/Observable.html

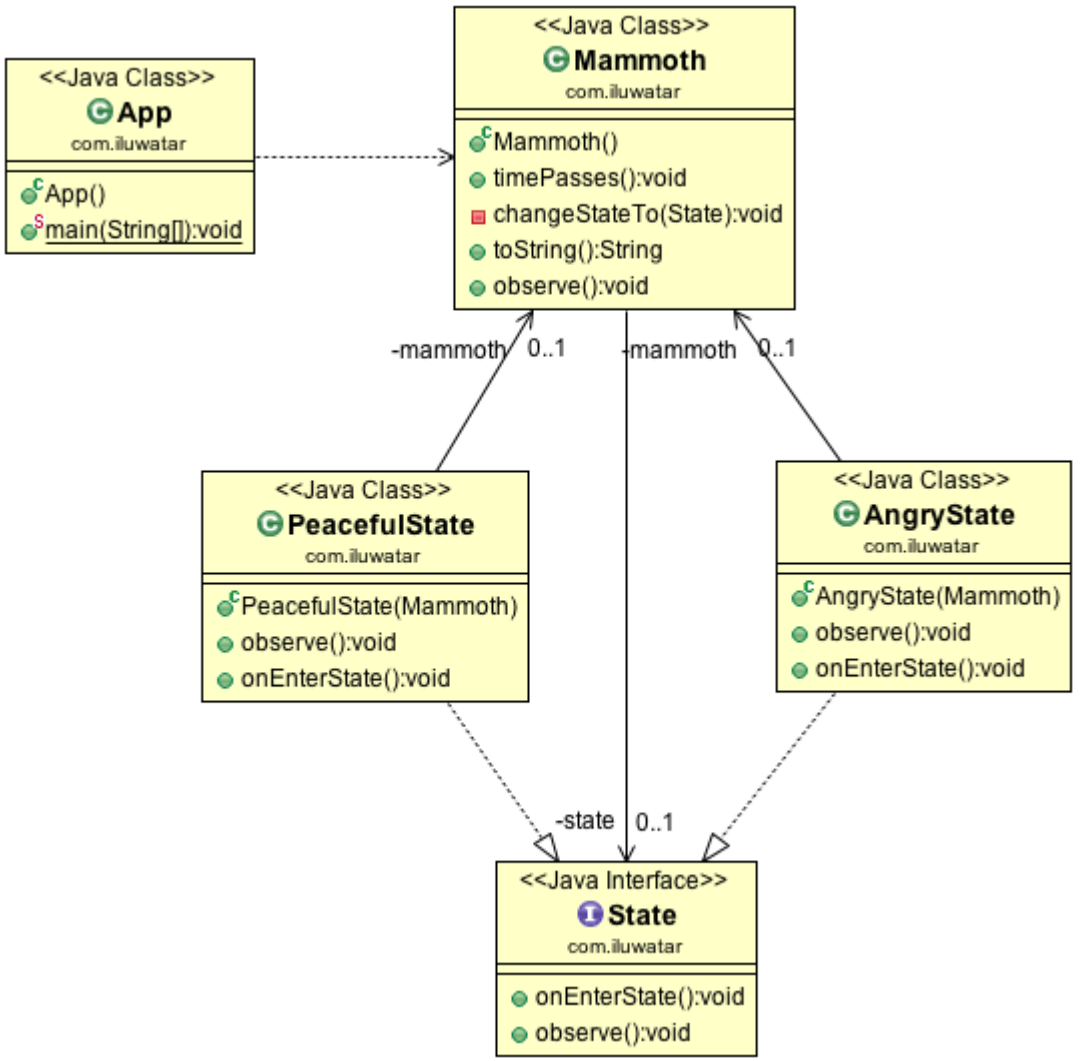
State (1)

- **Aim**:
  - It lets the object to change its behavior by changing its inner state.
  - Appearingly the object changes its class.

State (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/state

State (3)

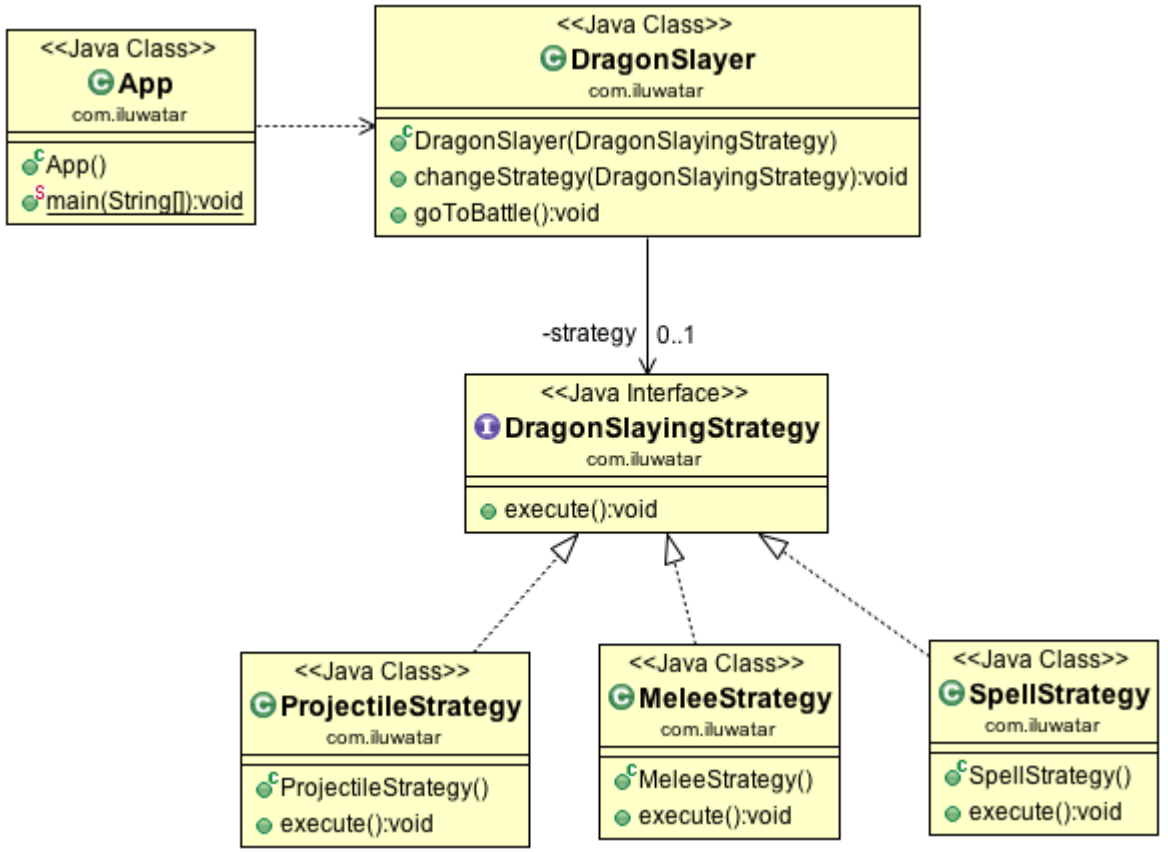

<<Java Class>>
App
com.iluwatar
App()
main(String[]):void
<<Java Class>>
Mammoth
com.iluwatar
Mammoth()
timePasses():void
changeStateTo(State):void
toString():String
observe():void
-mammoth
0..1
-mammoth
0..1
<<Java Class>>
PeacefulState
com.iluwatar
PeacefulState(Mammoth)
observe():void
onEnterState():void
<<Java Class>>
AngryState
com.iluwatar
AngryState(Mammoth)
observe():void
onEnterState():void
-state
0..1
<<Java Interface>>
State
com.iluwatar
onEnterState():void
observe():void

Strategy (1)

- **Aim**:
  - Defines an algorithm family in which the algorithms are collected to separate units making it able to replece them by each other
  - As a result the algorithm can be modified independently of the client

Strategy (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/strategy



Strategy (3)

- **Known uses**:
  - `java.util.Collections` https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html
    - See `binarySearch()`, `max()`, `min()` and `sort()` methods.
    - Related interfaces:
      - `java.lang.Comparable` https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html
      - `java.util.Comparator` https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html

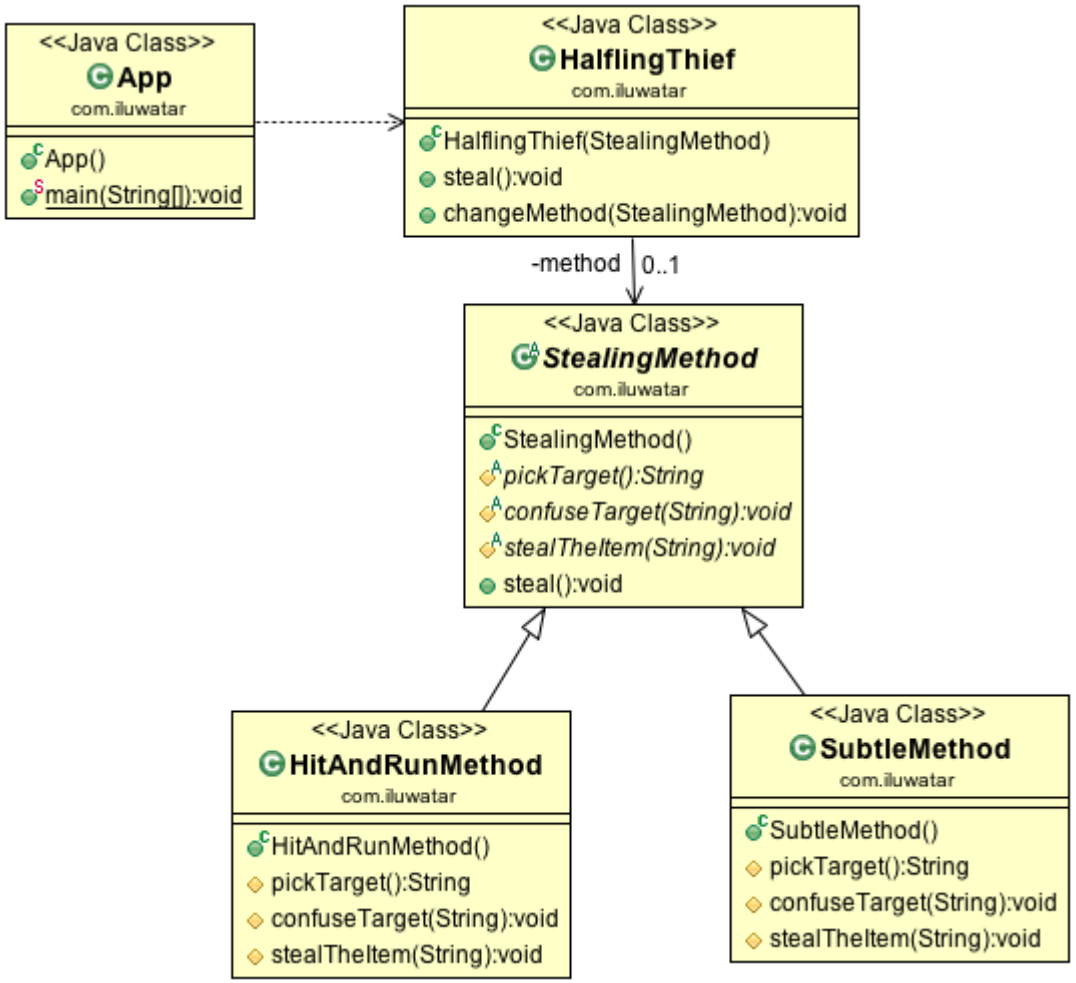
Template method (1)

- **Aim**
  - It provides the skeleton of the algorithm of a given method while letting the subclasses the implement the steps.
  - As a result the subclasses may override given steps of the algorithm without modifying the structure of it

Template method (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/template-method

Template method (3)



Template method (4)

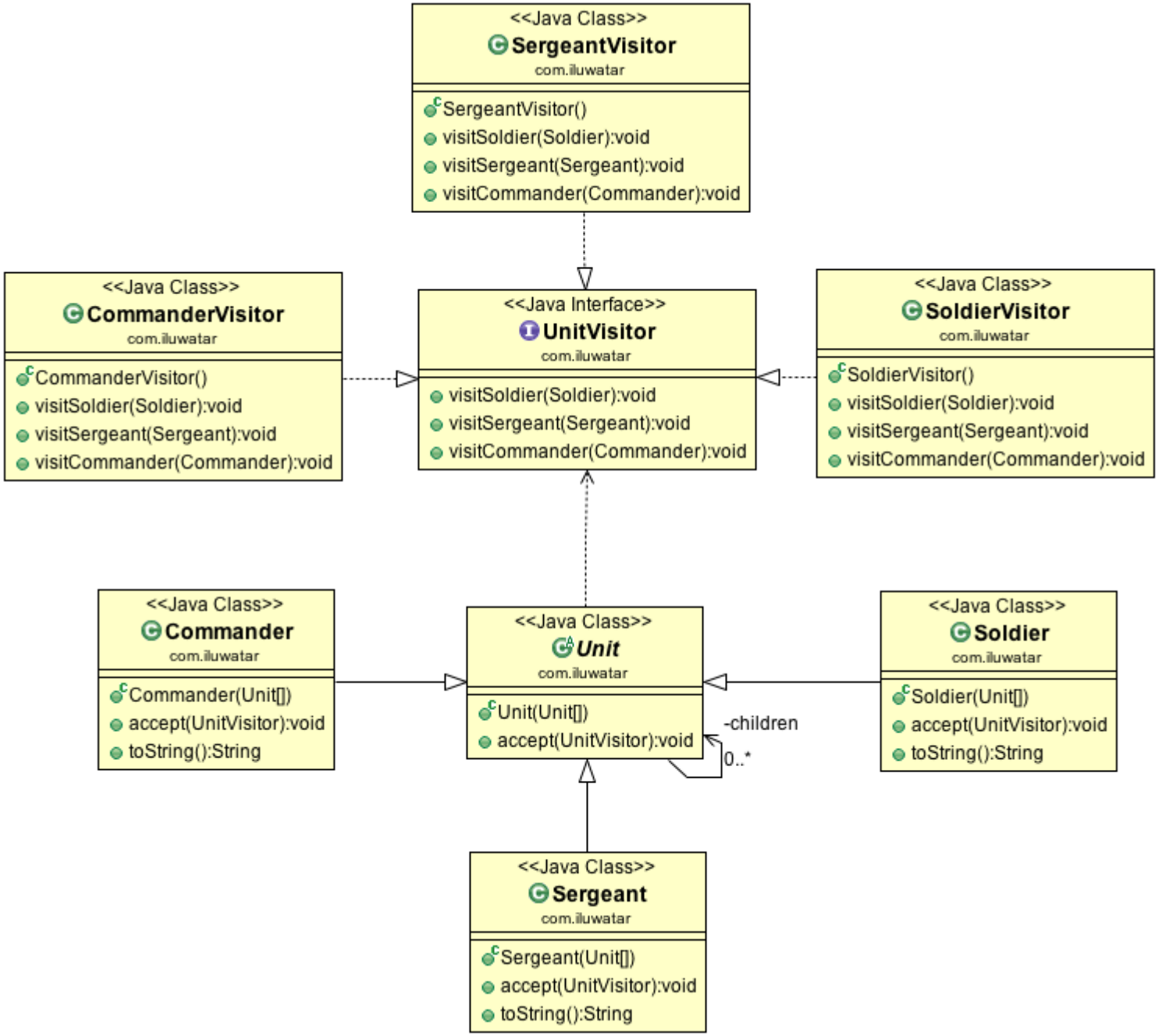
- **Known uses**:
  - `java.io.InputStream` https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html
  - `java.io.OutputStream` https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html
  - `java.util.AbstractList` https://docs.oracle.com/javase/8/docs/api/java/util/AbstractList.html
  - `java.util.AbstractMap` https://docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.html
  - `java.util.AbstractQueue` https://docs.oracle.com/javase/8/docs/api/java/util/AbstractQueue.html

Visitor (1)

- **Aim**:
  - Represents an operation that can be executed on the elements of an object structure
  - By the use of this pattern we can define a new operation without the need of changing the class of the affected elements

Visitor (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/visitor


<<Java Class>>
SergeantVisitor
com.iluwatar
SergeantVisitor()
visitSoldier(Soldier):void
visitSergeant(Sergeant):void
visitCommander(Commander):void
<<Java Class>>
CommanderVisitor
com.iluwatar
CommanderVisitor()
visitSoldier(Soldier):void
visitSergeant(Sergeant):void
visitCommander(Commander):void
<<Java Interface>>
UnitVisitor
com.iluwatar
visitSoldier(Soldier):void
visitSergeant(Sergeant):void
visitCommander(Commander):void
<<Java Class>>
SoldierVisitor
com.iluwatar
SoldierVisitor()
visitSoldier(Soldier):void
visitSergeant(Sergeant):void
visitCommander(Commander):void
<<Java Class>>
Commander
com.iluwatar
Commander(Unit[])
accept(UnitVisitor):void
toString():String
<<Java Class>>
Unit
com.iluwatar
Unit(Unit[])
accept(UnitVisitor):void
-children
0..*
<<Java Class>>
Soldier
com.iluwatar
Soldier(Unit[])
accept(UnitVisitor):void
toString():String
<<Java Class>>
Sergeant
com.iluwatar
Sergeant(Unit[])
accept(UnitVisitor):void
toString():String

Visitor (4)

- **Known uses**:
  - `java.nio.file.FileVisitor`
    https://docs.oracle.com/javase/8/docs/api/java/nio/file/FileVisitor.html
  - `javax.lang.model.element.Element`
    https://docs.oracle.com/javase/8/docs/api/javax/lang/model/element/Element.html
    `javax.lang.model.element.ElementVisitor`
    https://docs.oracle.com/javase/8/docs/api/javax/lang/model/element/ElementVisitor.html

More behavioral patterns

- *Null Object*
  https://github.com/iluwatar/java-design-patterns/tree/master/null-object
- *Servant*
  https://github.com/iluwatar/java-design-patterns/tree/master/servant
- *Specification*
  https://github.com/iluwatar/java-design-patterns/tree/master/specification
- ...

Null object (1)

- **Aim**:
  - Provide an alternative of the `null` reference to indicate the absence of an object by the use of an object that implements the required interface by providing only empty method bodies for the abstract methods.

Null object (2)

- **Example code**: https://github.com/iluwatar/java-design-patterns/tree/master/null-object