

Agile Software Development

Produced
by

Eamonn de Leastar (edelestar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRCE

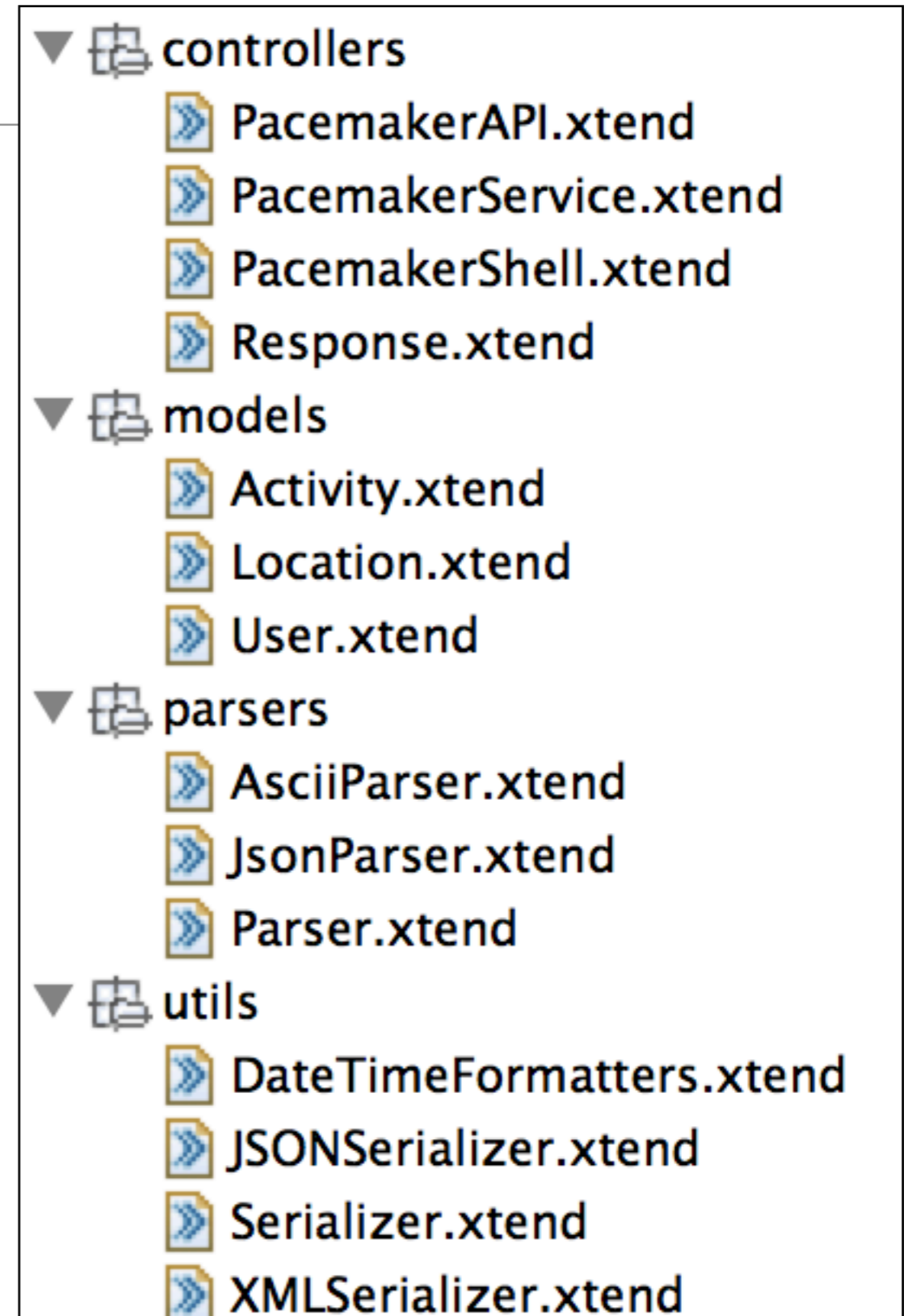
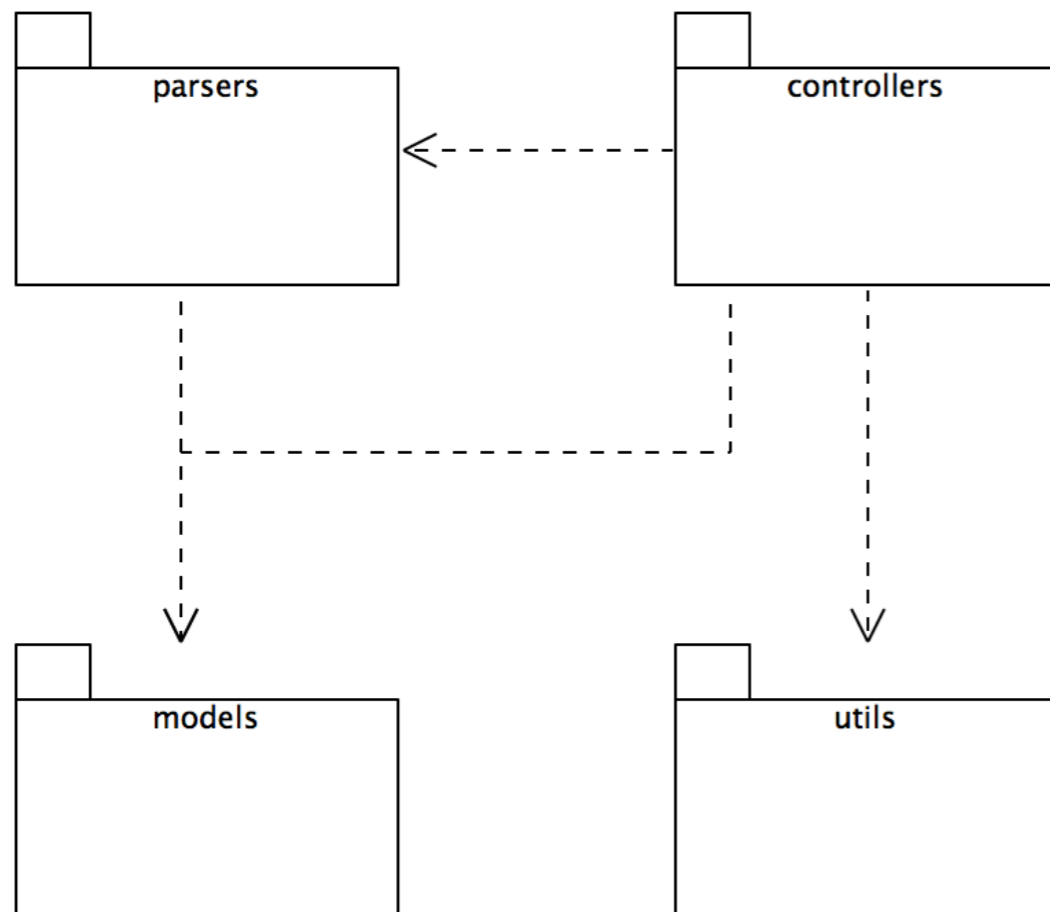


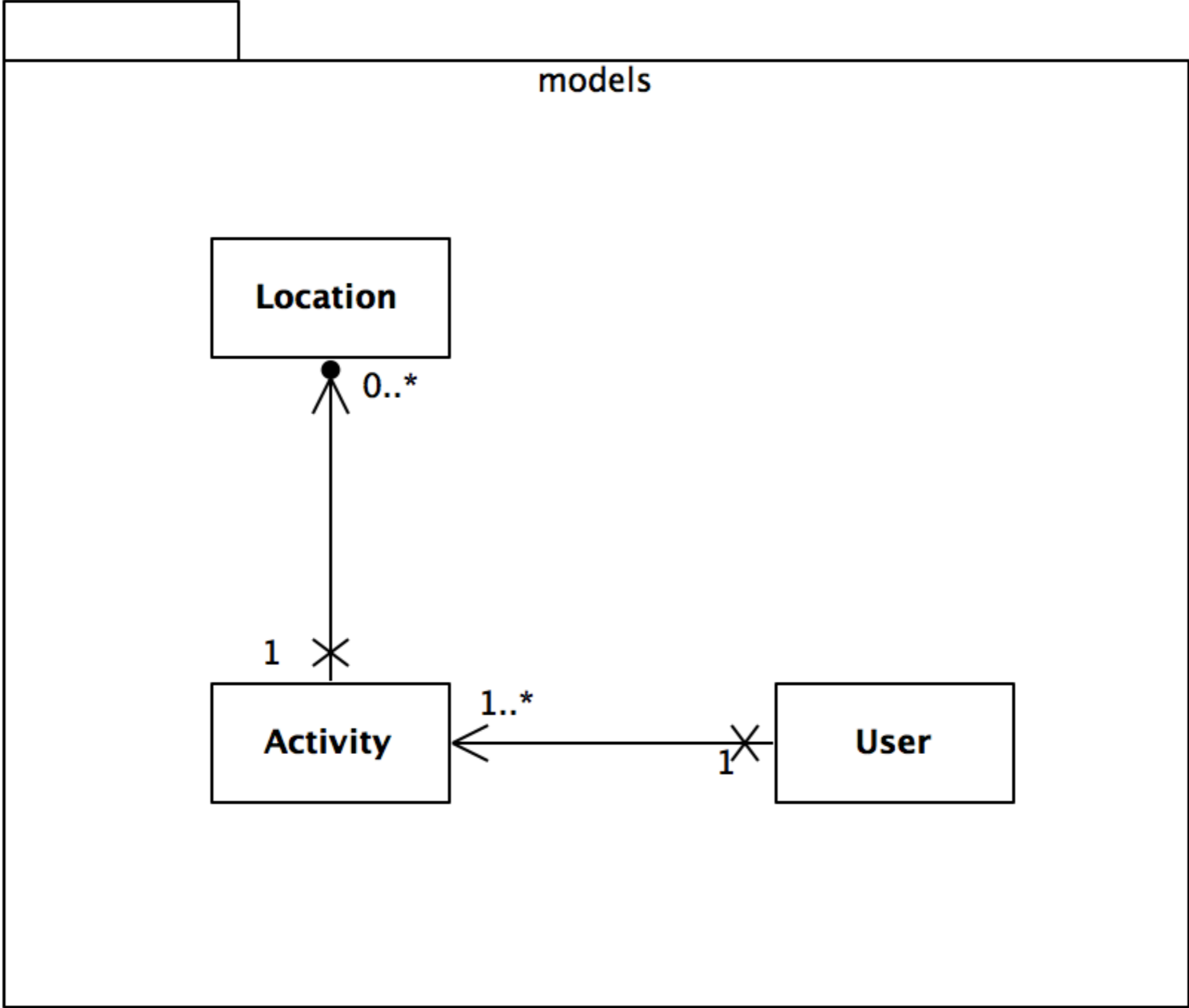
Assignment Solution

UML + Code

pacemaker-console-maven

- uml package diagram



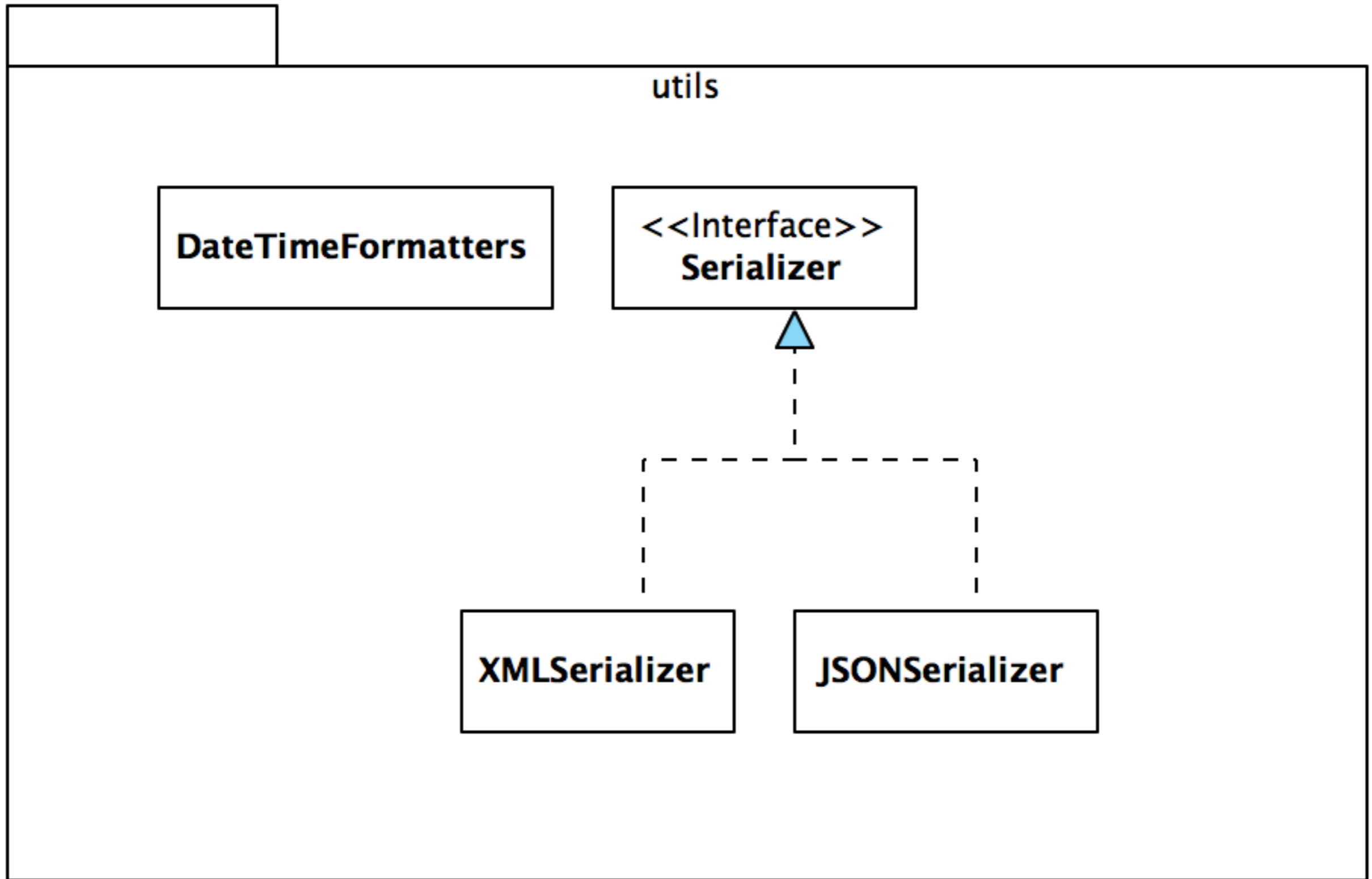


models

```
@Data class User
{
    Long    id
    String  firstname
    String  lastname
    String  email
    String  password
    @Property Map<Long, Activity> activities = new HashMap
}
```

```
@Data class Activity
{
    Long    id
    String  type
    String  location
    double  distance
    DateTime starttime
    Duration duration
    @Property List<Location> route = new ArrayList
}
```

```
@Data class Location
{
    float latitude
    float longitude
}
```



utils

```
interface Serializer
{
    def void push(Object o)
    def Object pop()
    def void write()
    def void read()
}
```

```

class DateTimeFormatters
{
  static val periodFormatter = new PeriodFormatterBuilder().printZeroAlways()
                                                                    .appendHours()
                                                                    .appendSeparator(":")
                                                                    .appendMinutes()
                                                                    .appendSeparator(":")
                                                                    .appendSeconds()
                                                                    .toFormatter();

  static val dateFormatter = DateTimeFormat.forPattern("dd:MM:yyyy HH:mm:ss");

  def static parseDateTime (String dateTime)
  {
    new DateTime(dateFormatter.parseDateTime(dateTime))
  }

  def static parseDateTime (DateTime dateTime)
  {
    dateFormatter.print(dateTime)
  }

  def static parseDuration (String duration)
  {
    periodFormatter.parsePeriod(duration).toStandardDuration
  }

  def static parseDuration (Duration duration)
  {
    periodFormatter.print(duration.toPeriod)
  }
}

```



```

class XMLSerializer implements Serializer
{
    var Deque<Object> stack = new ArrayDeque
    val File file;

    new (String filename)
    {
        this.file = new File(filename + '.xml');
    }

    def override void push(Object o)
    {
        stack.push(o)
    }

    def override Object pop()
    {
        return stack.pop();
    }

    @SuppressWarnings("unchecked")
    def override void read()
    {
        var ObjectInputStream is = null

        try
        {
            val xstream = new XStream(new DomDriver())
            is = xstream.createObjectInputStream(
                new FileReader(file))

            stack = is.readObject as Deque<Object>
        }
        finally
        {
            if (is != null)
            {
                is.close();
            }
        }
    }
}

```

```

def override void write()
{
    var ObjectOutputStream os = null

    try
    {
        val xstream = new XStream(new DomDriver())
        os = xstream.createObjectOutputStream
            (new FileWriter(file))

        os.writeObject(stack)
    }
    finally
    {
        if (os != null)
        {
            os.close
        }
    }
}

```

```
class JsonSerializer implements Serializer
{
    // ...

    @SuppressWarnings("unchecked")
    def override void read()
    {
        var ObjectInputStream is = null

        try
        {
            val xstream = new XStream(new JettisonMappedXmlDriver())
            is = xstream.createObjectInputStream(new FileReader(file))
            stack = is.readObject as Deque<Object>
        }
        finally
        {
            if (is != null)
            {
                is.close();
            }
        }
    }
}
```

```
def override void write()
{
    var ObjectOutputStream os = null

    try
    {
        val xstream = new XStream(new JettisonMappedXmlDriver())
        os = xstream.createObjectOutputStream(new FileWriter(file))
        os.writeObject(stack)
    }
    finally
    {
        if (os != null)
        {
            os.close
        }
    }
}
```

parsers

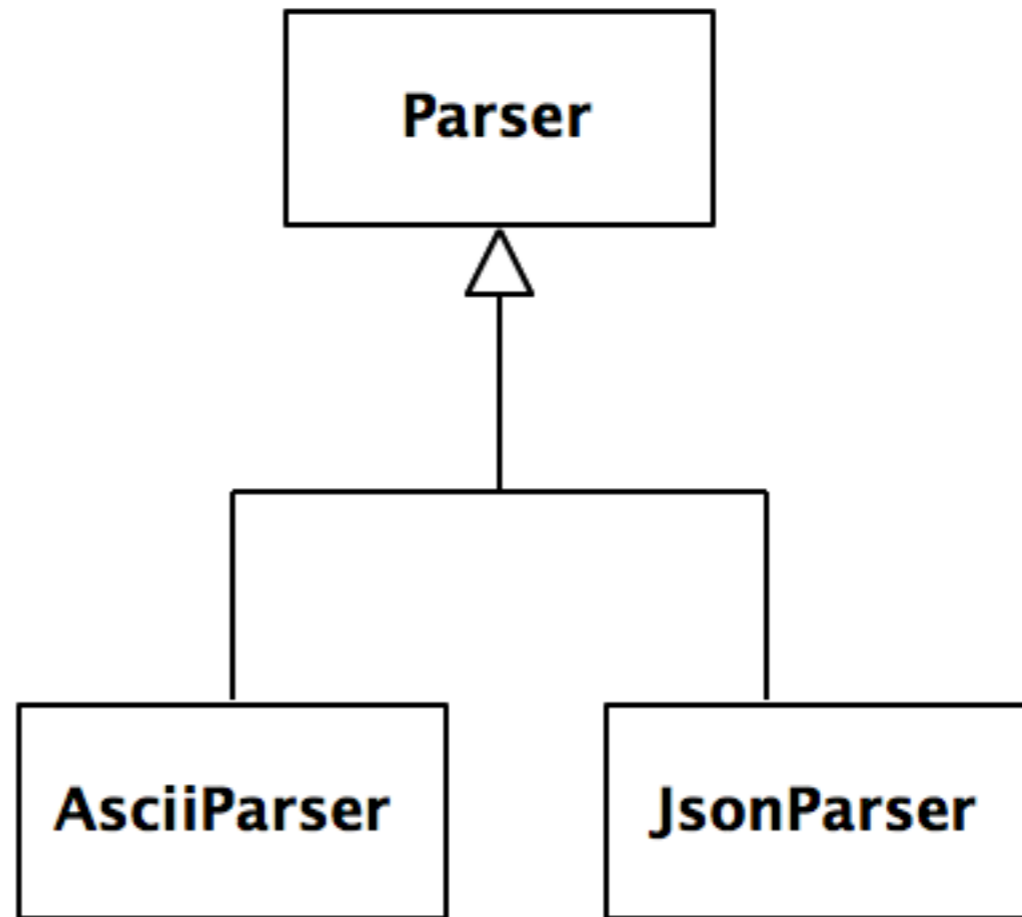
<<Interface>>
UserMixin

<<Interface>>
ActivityMixin

Parser

AsciiParser

JsonParser



```
class Parser
{
  def String renderUser(User user)
  {
    user.toString
  }

  def String renderUsers(Collection<User> users)
  {
    users.toString
  }

  def String renderActivities(Collection<Activity> activities)
  {
    activities.toString
  }
}
```

```

class AsciiParser extends Parser
{
  override renderUser(User user)
  {
    val List<User> userList = new ArrayList()
    userList.add(user)
    renderUsers(userList)
  }

  override renderActivities(Collection<Activity> activities)
  {
    if (!activities.empty)
    {
      val List<Activity> activityList = new ArrayList(activities)
      var activitiesTable = new CollectionASCIITableAware<Activity>
        (activityList, "id", "type", "location", "distance", "starttime", "duration", "route")
      ASCIIITable.getInstance().getTable(activitiesTable)
    }
  }

  override renderUsers(Collection<User> users)
  {
    if (!users.empty)
    {
      val List<User> userList = new ArrayList(users)
      var asciiTableAware = new CollectionASCIITableAware<User>
        (userList, "id", "firstname", "lastname", "email", "password")
      ASCIIITable.getInstance().getTable(asciiTableAware);
    }
  }
}

```

```

interface UserMixin
{
    @JsonIgnore def String getActivities()
    @JsonIgnore def Long getId()
}

interface ActivityMixin
{
    @JsonIgnore def Long getId()
}

class JsonParser extends Parser
{
    val mapper = new ObjectMapper()

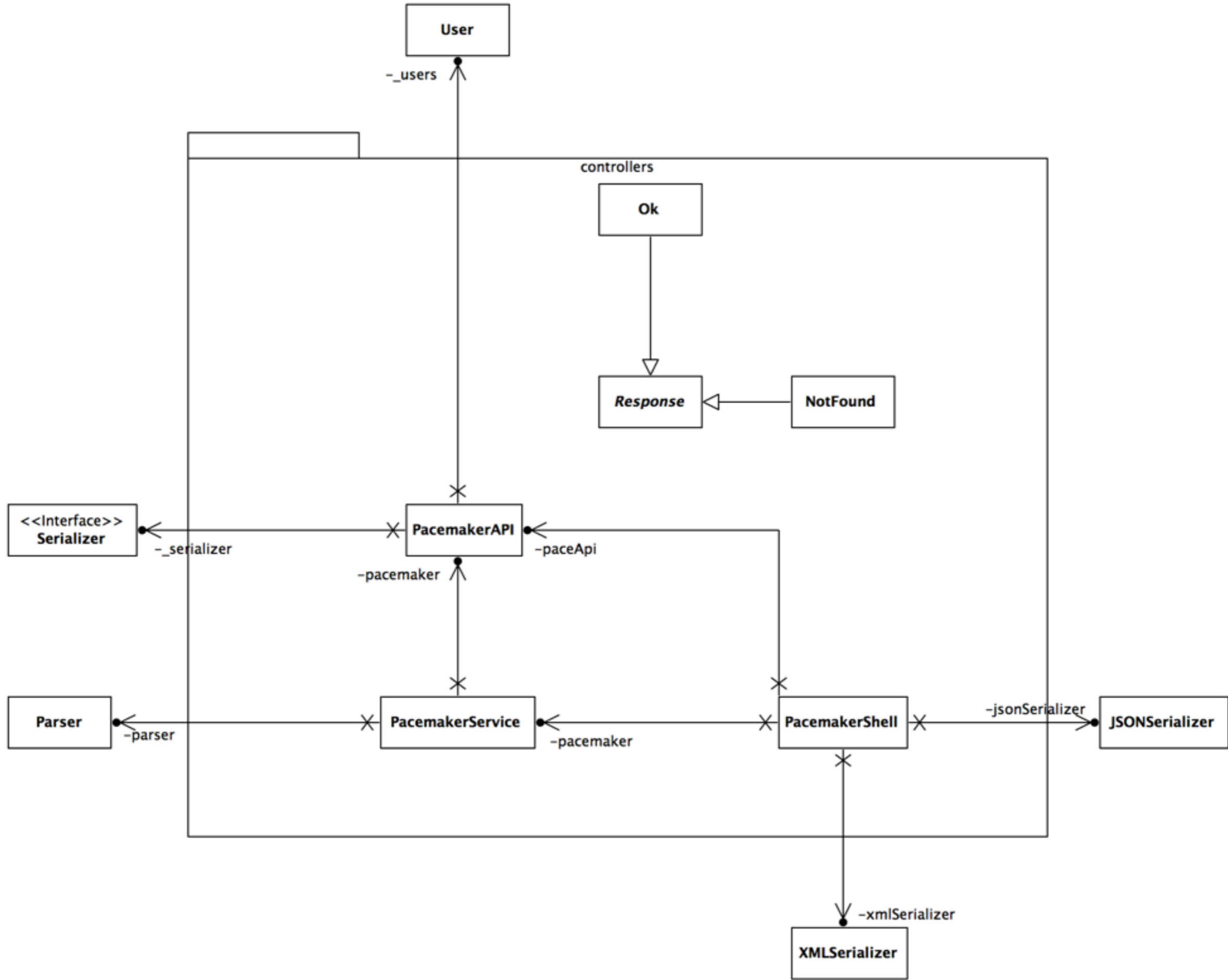
    new()
    {
        mapper.addMixInAnnotations(typeof(User),    typeof(UserMixin))
        mapper.addMixInAnnotations(typeof(Activity), typeof(ActivityMixin))
    }

    override renderUser(User user)
    {
        mapper.writerWithDefaultPrettyPrinter.writeValueAsString(user)
    }

    override renderActivities(Collection<Activity> activities)
    {
        if (activities.size > 0)
        {
            mapper.writerWithDefaultPrettyPrinter.writeValueAsString(activities)
        }
    }

    override renderUsers(Collection<User> users)
    {
        mapper.writerWithDefaultPrettyPrinter.writeValueAsString(users)
    }
}

```



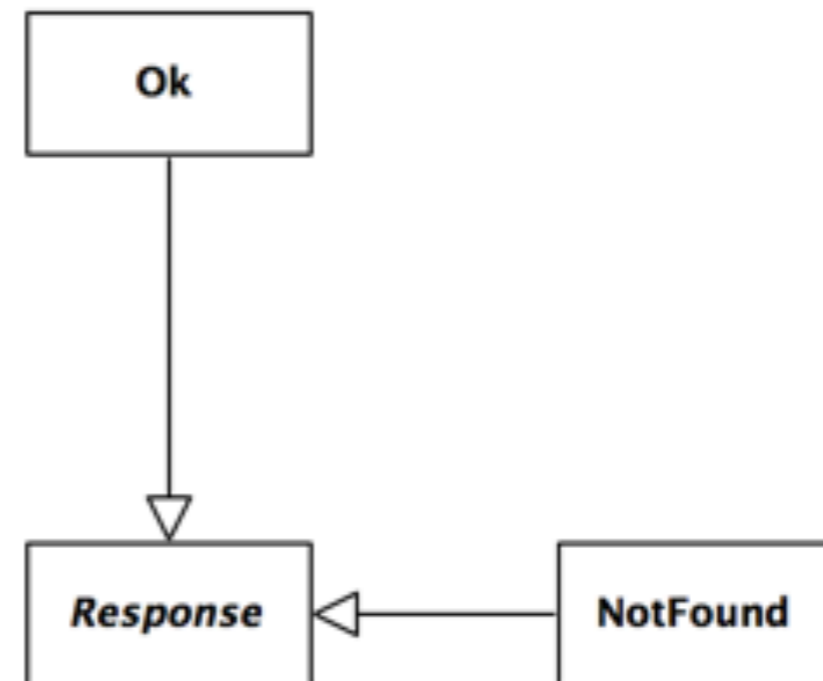
```
abstract class Response
{
    var String response

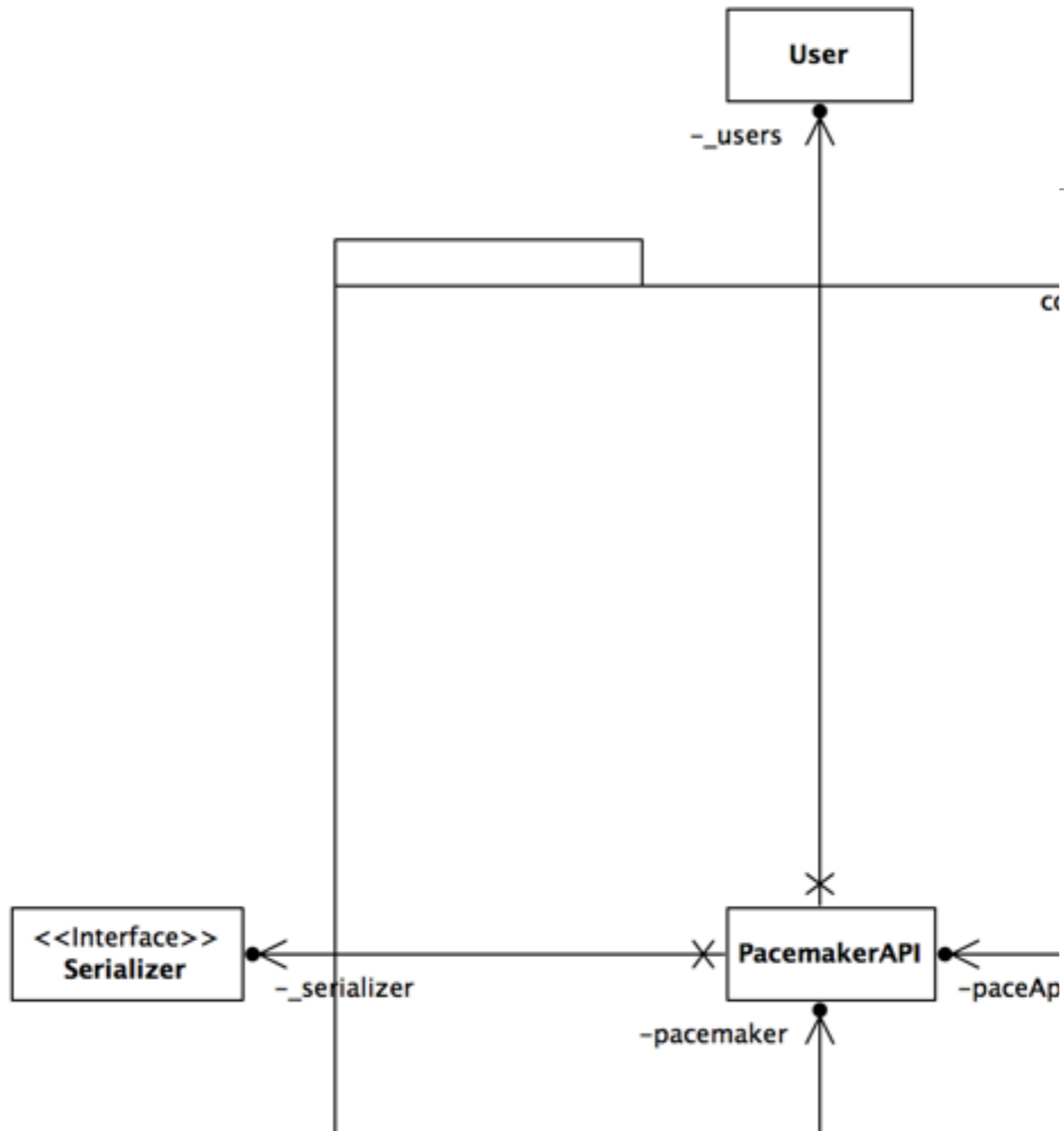
    new(String response)
    {this.response = response }

    override toString()
    { response }
}

class Ok extends Response
{
    new (String response)
    {super("ok\n" + response)}
}

class NotFound extends Response
{
    new (String response)
    {super("not found\n")}
}
```





```

class PacemakerAPI
{
    static long userIndex = 0;
    static long activityIndex = 0;

    var Map<Long, User>      userMap      = new HashMap
    var Map<String, User>    userEmailMap = new HashMap
    var Map<Long, Activity> activityMap   = new HashMap
    @Property Collection<User> users      = userMap.values
    @Property Serializer     serializer

    new()
    {
        userIndex = 0
        activityIndex = 0
    }

    def void load() throws Exception
    {
        serializer.read();
        activityIndex = serializer.pop() as Long
        userIndex     = serializer.pop() as Long
        activityMap   = serializer.pop() as Map<Long, Activity>
        userEmailMap = serializer.pop() as Map<String, User>
        userMap       = serializer.pop() as Map<Long, User>
        users         = userMap.values
    }

    def void store()
    {
        serializer.push(userMap)
        serializer.push(userEmailMap)
        serializer.push(activityMap)
        serializer.push(userIndex)
        serializer.push(activityIndex)
        serializer.write()
    }
}

```

```
def Long createUser (String firstName, String lastName, String email, String password)
{
    userIndex = userIndex + 1
    var user = new User (userIndex, firstName, lastName, email, password)
    userMap.put(userIndex, user);
    userEmailMap.put(user.email, user)
    userIndex
}

def getUser (Long id)
{
    userMap.get(id)
}

def getUser (String email)
{
    userEmailMap.get(email)
}

def deleteUser (Long id)
{
    userEmailMap.remove(userMap.get(id))
    userMap.remove(id)
}

def deleteUser (String email)
{
    val user = userEmailMap.remove(getUser(email))
    userMap.remove(user.id)
}
```

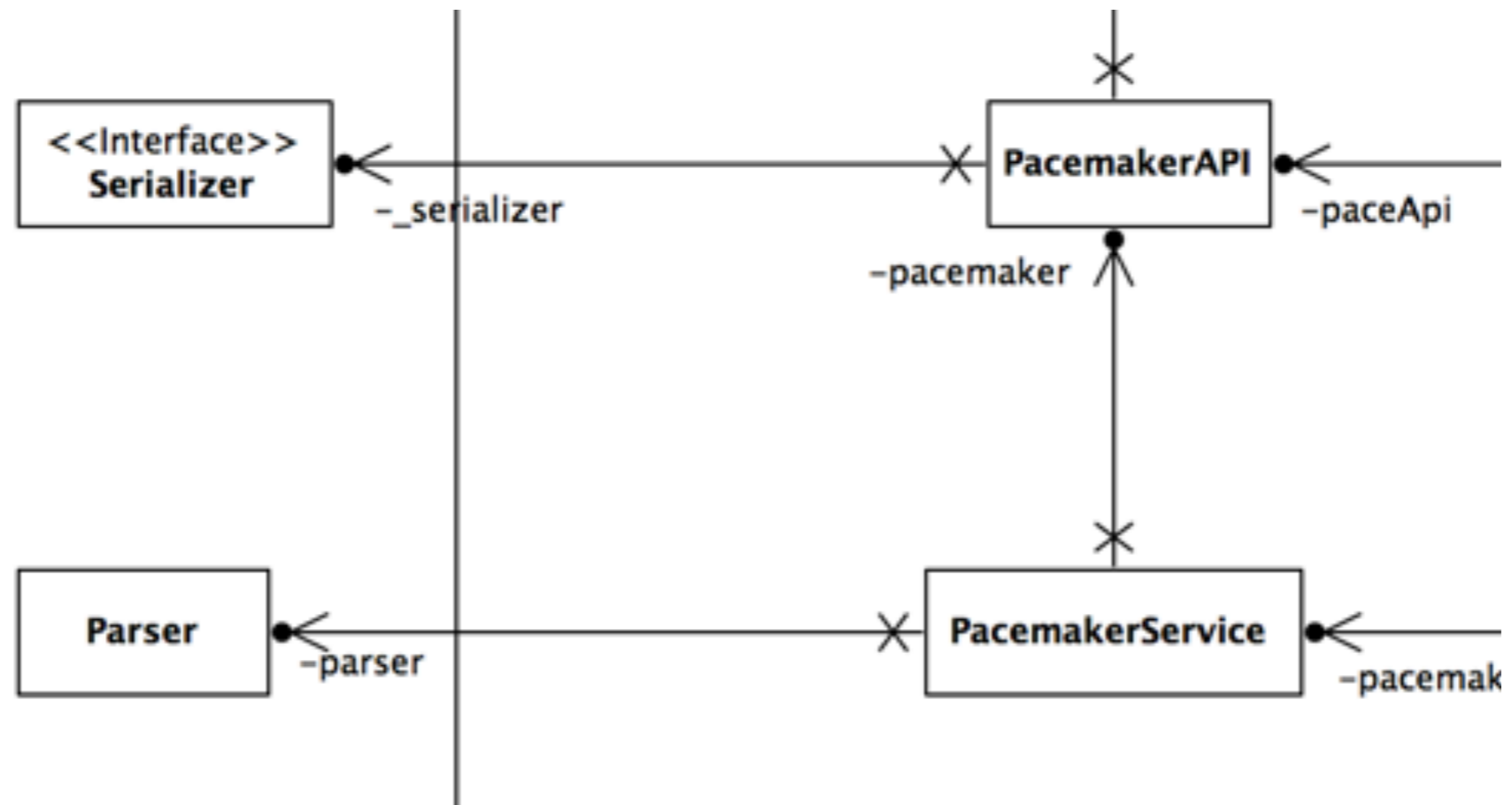
```

def createActivity(Long id, String type, String location, double distance,
                  String dateStr, String durationStr)
{
    var Activity activity = null;
    var user = Optional.fromNullable(userMap.get(id))
    if (user.isPresent())
    {
        activityIndex = activityIndex + 1
        activity = new Activity (activityIndex, type, location, distance,
                               parseDateTime(dateStr), parseDuration(durationStr))
        user.get.activities.put(activity.id, activity);
        activityMap.put(activity.id, activity);
    }
    return activity;
}

def getActivity (Long id)
{
    activityMap.get(id)
}

def void addLocation (Long id, float latitude, float longitude)
{
    val activity = Optional.fromNullable(activityMap.get(id))
    if (activity.isPresent())
    {
        activity.get.route.add(new Location(latitude, longitude));
    }
}

```



```

class PacemakerService
{
    var PacemakerAPI pacemaker
    var Parser      parser

    new(PacemakerAPI pacemaker, Parser parser)
    {
        this.pacemaker = pacemaker
        this.parser     = parser
    }

    def createUser(String firstname, String lastname, String email, String password)
    {
        val id = pacemaker.createUser(firstname, lastname, email, password)
        new Ok(parser.renderUser(pacemaker.getUser(id)))
    }

    def getUser(Long id)
    {
        val user = pacemaker.getUser(id)
        if (null != user) new Ok(parser.renderUser(user)) else new NotFound("")
    }

    def getUser(String email)
    {
        val user = pacemaker.getUser(email)
        if (null != user) getUser(user.id) else new NotFound("")
    }

    def getUsers()
    {
        new Ok(parser.renderUsers(pacemaker.users))
    }

    def deleteUser(Long id)
    {
        val user = pacemaker.getUser(id)
        pacemaker.deleteUser(user?.id)
        if (null != user) new Ok("") else new NotFound("")
    }
}

```

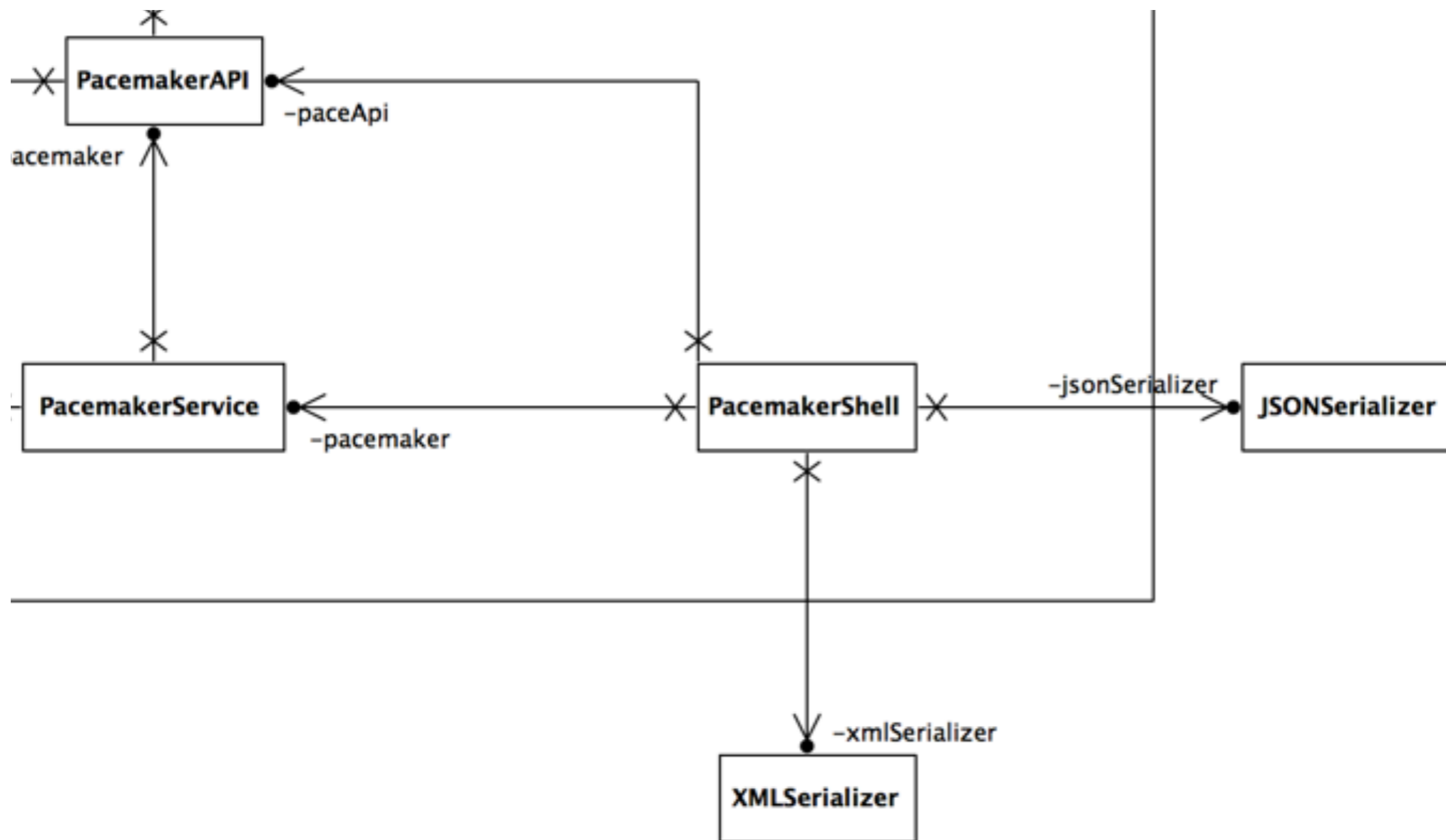
```
def createActivity(Long id, String type, String location, double distance, String dateStr, String durationStr)
{
  if (null != pacemaker.getUser(id))
  {
    pacemaker.createActivity(id, type, location, distance, dateStr, durationStr)
    new Ok("")
  }
  else new NotFound("")
}

def getActivities(Long id)
{
  val user = pacemaker.getUser(id)
  if (null != user) new Ok(parser.renderActivities(user.activities.values)) else new NotFound("")
}

def addLocation (Long id, float latitude, float longitude)
{
  val activity = pacemaker.getActivity(id)
  if (null != activity)
  {
    pacemaker.addLocation(id, latitude, longitude)
    new Ok("")
  }
  else new NotFound("")
}
```

```
def listActivities (Long id, String sortBy)
{
  val activities = pacemaker.getUser(id).activities.values

  val report = switch (sortBy)
  {
    case "type"      : activities.sortBy[type]
    case "location"  : activities.sortBy[location]
    case "distance"  : activities.sortBy[distance]
    case "date"      : activities.sortBy[starttime]
    case "duration"  : activities.sortBy[duration]
  }
  return new Ok(parser.renderActivities(report))
}
```

```
class PacemakerShell
{
    var PacemakerAPI      paceApi
    var PacemakerService  pacemaker
    val datastore         = "testdatastore";
    val xmlSerializer     = new XMLSerializer(datastore);
    val jsonSerializer    = new JsonSerializer(datastore);

    new()
    {
        paceApi = new PacemakerAPI;
        pacemaker = new PacemakerService (paceApi, new AsciiParser as Parser)
    }

    @Command(description="List all users details")
    def void listUsers ()
    {
        println(pacemaker.getUsers)
    }

    @Command(description="Create a new User")
    def void createUser (@Param(name="first name") String firstname, @Param(name="last name") String lastname,
                        @Param(name="email") String email, @Param(name="password") String password)
    {
        println (pacemaker.createUser(firstname, lastname, email, password))
    }

    @Command(description="List a users details")
    def void listUser (@Param(name="email") String email)
    {
        println (pacemaker.getUser(email))
    }

    @Command(description="List a users details")
    def void listUser (@Param(name="id") Long id)
    {
        println (pacemaker.getUser(id))
    }
}
```

```

@Command(description="List a users activities")
def void listActivities (@Param(name="user id") Long id)
{
    println (pacemaker.getActivities(id))
}

@Command(description="Delete a User")
def void deleteUser (@Param(name="id") Long id)
{
    println (pacemaker.deleteUser(id))
}

@Command(description="Add an activity")
def void addActivity (@Param(name="user-id") Long id, @Param(name="type") String type,
                    @Param(name="location") String location, @Param(name="distance") double distance,
                    @Param(name="datetime") String dateStr, @Param(name="duration") String durationStr
                    )
{
    try
    {
        println (pacemaker.createActivity(id, type, location, distance, dateStr, durationStr))
    }
    catch (IllegalArgumentException e)
    {
        println ("Date or Duration format error: " + e.message)
    }
}

@Command(description="Add Location to an activity")
def void addLocation (@Param(name="activity-id") Long id,
                    @Param(name="latitude") float latitude, @Param(name="longitude") float longitude)
{
    println (pacemaker.addLocation(id, latitude, longitude))
}

```

```
@Command(description="List Activities")
def void listActivities (@Param(name="userid") Long id,
                        @Param(name="sortBy: type, location, distance, date, duration") String sortBy)
{
    val options = #{"type", "location", "distance", "date", "duration"}
    if (options.contains(sortBy))
    {
        println (pacemaker.listActivities(id, sortBy))
    }
    else
        println ("usage : la " + options.toString)
}
```

```

@Command(description="Set file format")
def void changeFileFormat (@Param(name="file format: xml, json") String fileFormat)
{
    switch (fileFormat)
    {
        case 'xml' : paceApi.serializer = xmlSerializer
        case 'json' : paceApi.serializer = jsonSerializer
    }
}

@Command(description="Load activities persistent store")
def void load ()
{
    paceApi.load
}

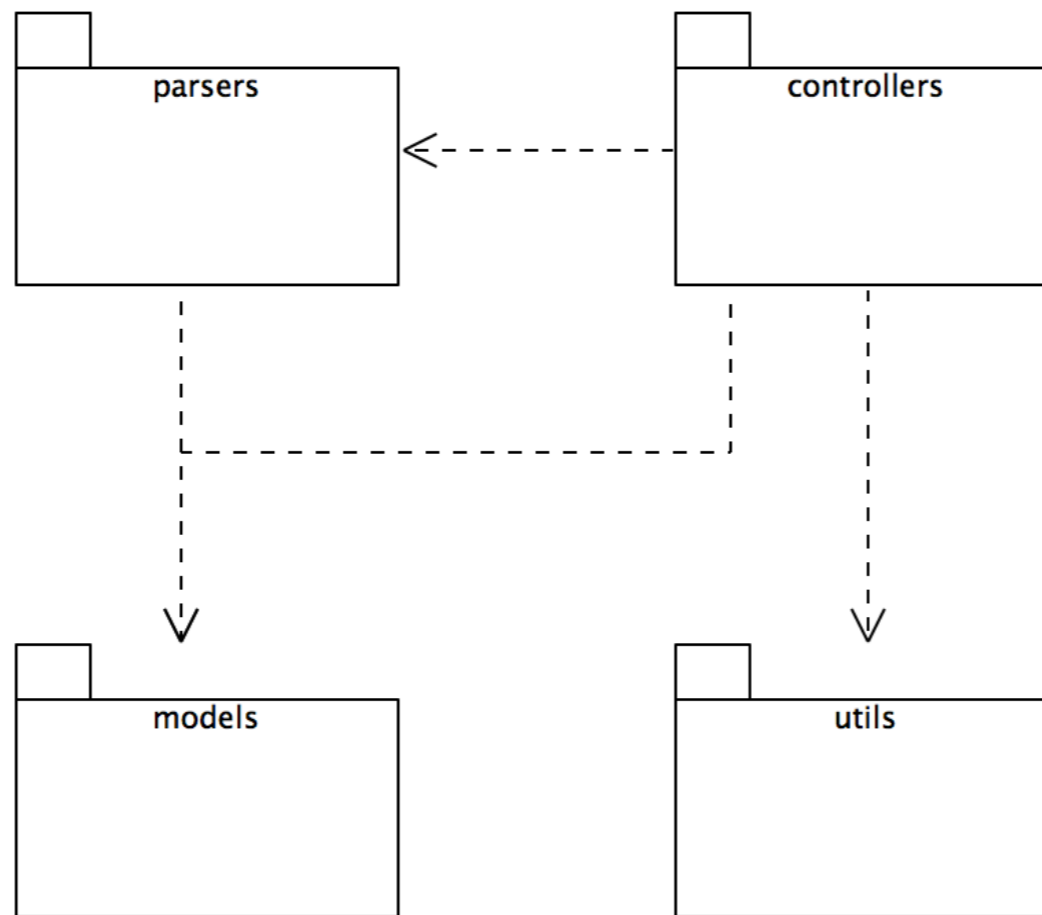
@Command(description="Store activities persistent store")
def void store ()
{
    paceApi.store
}

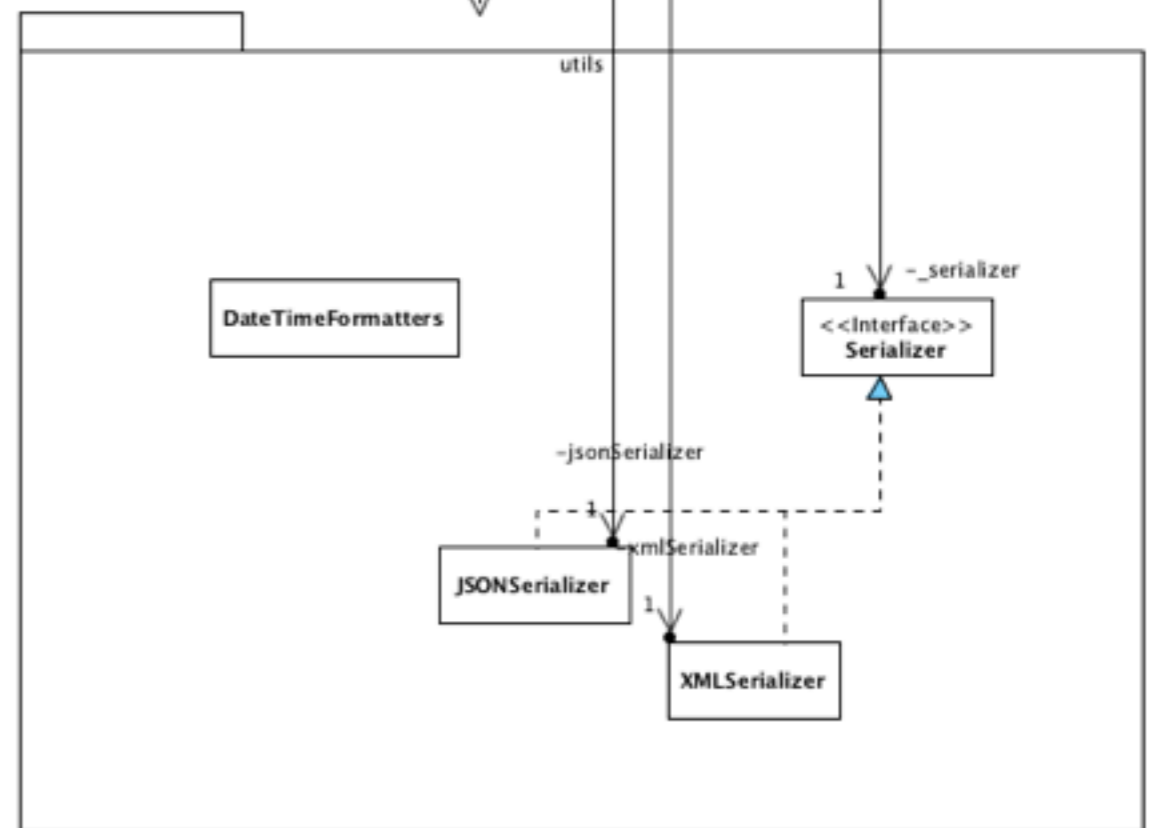
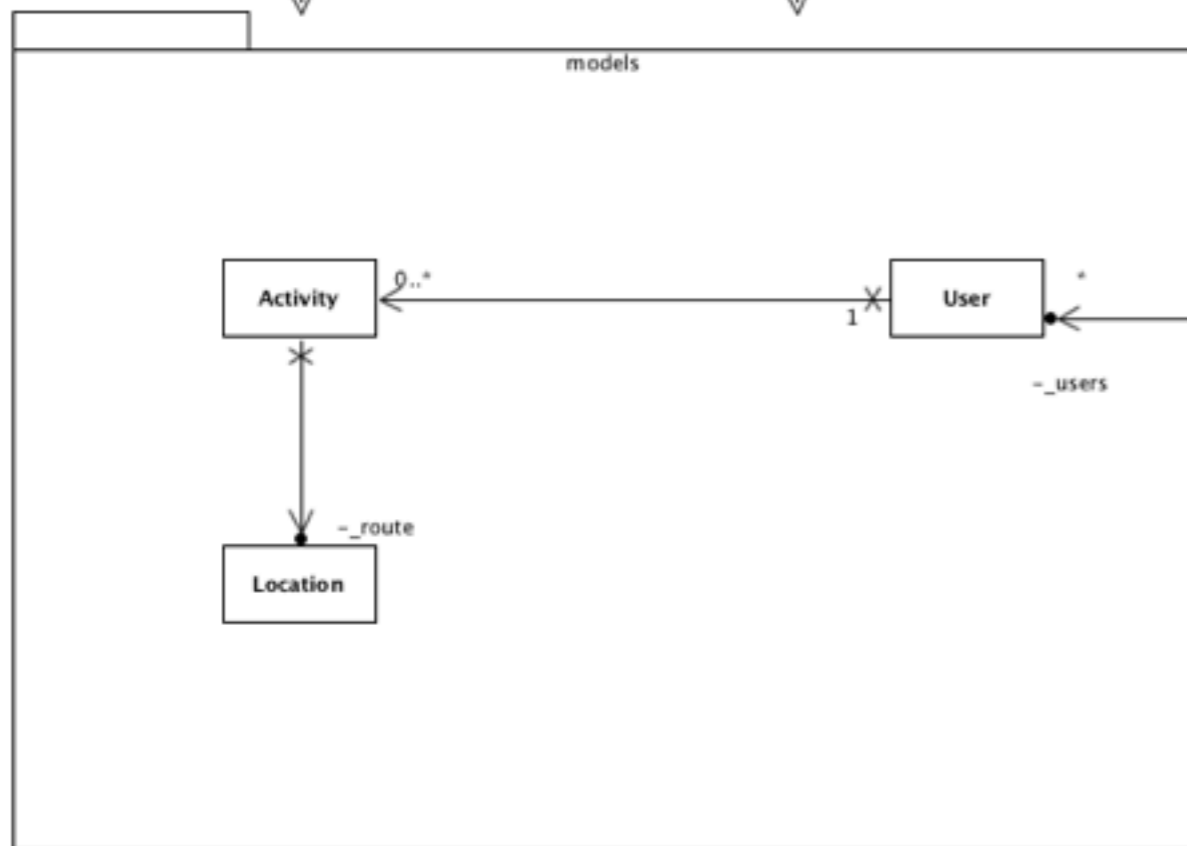
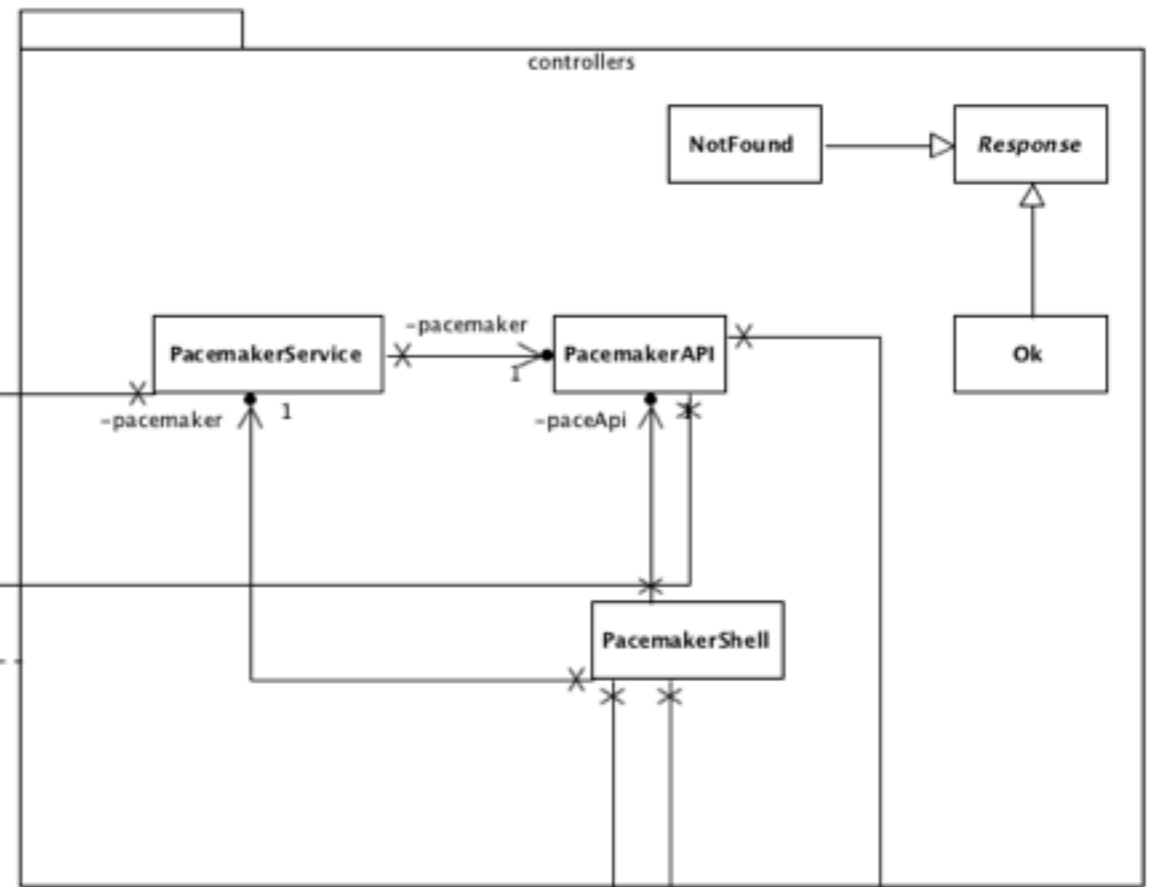
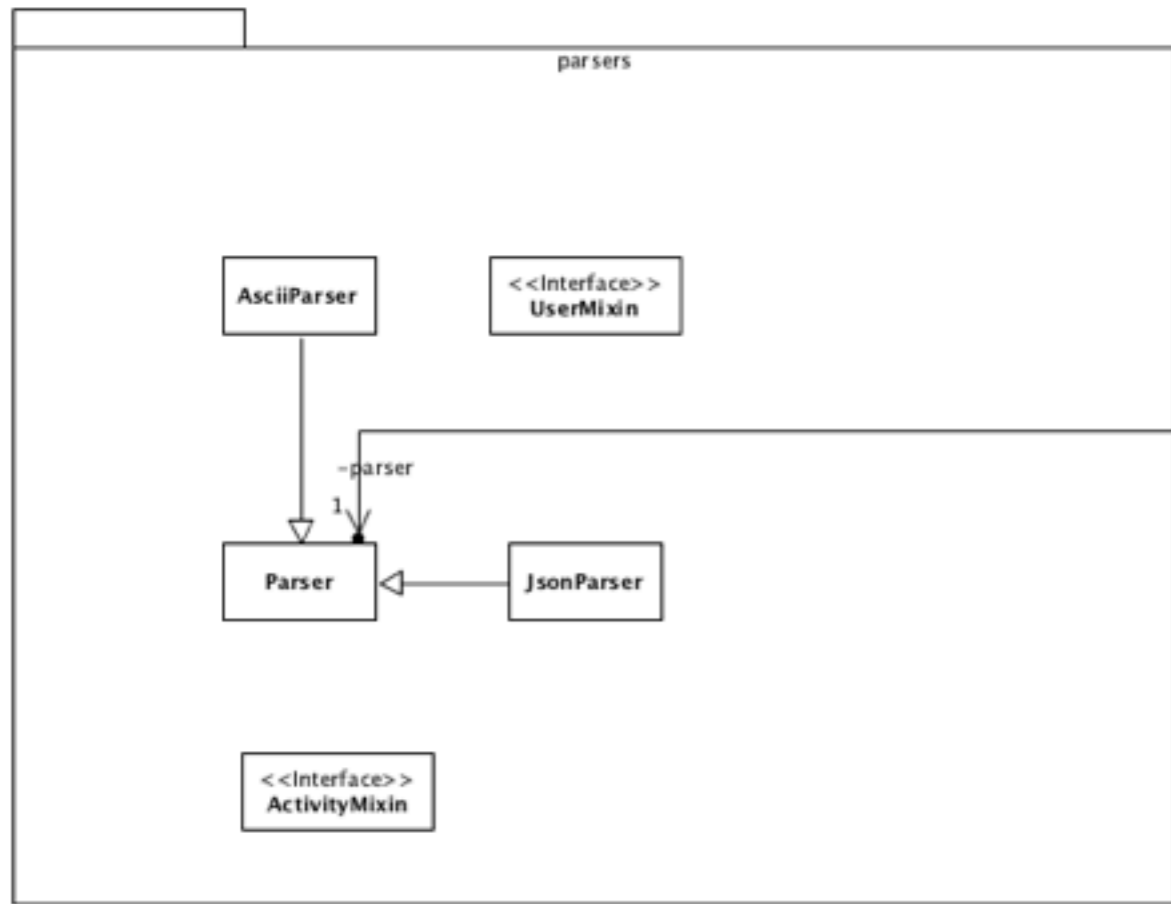
def static void main(String[] args) throws Exception
{
    val main = new PacemakerShell()

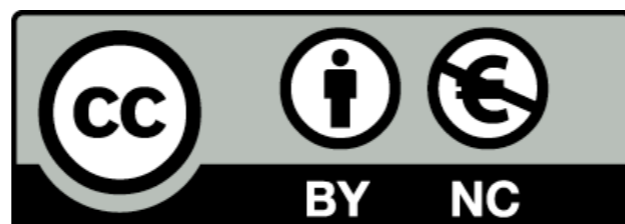
    val shell = ShellFactory.createConsoleShell("pm", "Welcome to pacemaker-console - ?help for instructions", main)
    shell.commandLoop

    main.paceApi.store
}

```







Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

