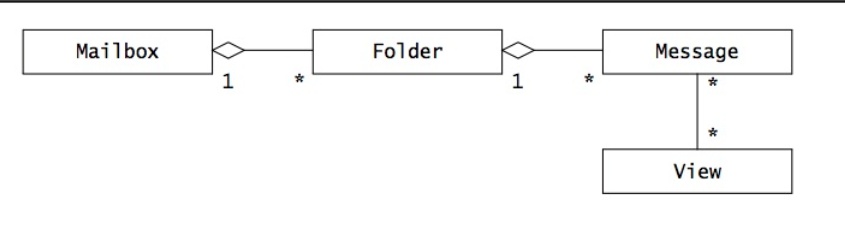
1. Apply appropriate transformations to the figure below. Show class declarations including attributes and method headers. Method body may be omitted.



**Since no ordering was specified on the associations, a Set was used to represent the many side. If you used a List, additional checking code would need to be added to ensure that multiple associations to the same object are not created.**

**public class Mailbox**

**{**

**Private Set folders = new HashSet();**

**public Folder[ ] getFolders() { return (Folder[])folders.toArray(); }**

**public void addFolder(Folder f) { folders.add(f);**

**f.setMailbox(this); }**

**public void removeFolder(Folder f) { folders.remove(f);**

**f.setMailbox(null); }**

**}**

**public class Folder**

**{**

**Private Mailbox mailbox = null;**

**Private Set messages = new HashSet();**

**public Mailbox getMailbox() { return mailbox; }**

**public void setMailbox(Mailbox newMailbox)**

**{**

**if (newMailbox != mailbox)**

**{**

**Mailbox oldMailbox = mailbox;**

**mailbox = newMailbox;**

**if (newMailbox != null) { newMailbox.addFolder(this); }**

**if (oldMailbox != null) { oldMailbox.removeFolder(this); }**

**}**

**}**

**public Message [] getMessages()**

**{**

**return (Message[])messages.toArray();**

**}**

**public void addMessage(Message m) { messages.add(m);**

**m.setFolder(this); }**

**public void removeMessage(Message m) { messages.remove(m);**

**m.setFolder(null); }**

**}**

**public class Message**

**{**

**Private Folder folder = null;**

**private Set views = new HashSet();**

**public Folder getFolder() { return folder; }**

**public void setFolder(Folder newFolder)**

**{**

**if (newFolder != folder)**

**{**

**Folder oldFolder = folder;**

**folder = newFolder;**

**if (newFolder != null) { newFolder.addMessage(this); }**

**if (oldFolder != null) { oldFolder.removeMessage(this); }**

**}**

**}**

**public View [] getViews() { return (View[])views.toArray(); }**

**public void addView(View v)**

**{**

**if (!views.contains(v)) { views.add(v);**

**v.addMessage(this); }**

**}**

**public void removeView(View v)**

**{**

**if (views.contains(v)) { views.remove(v);**

**v.removeMessage(this); }**

**}**

**}**

**public class View**

**{**

**private Set messages = new HashSet();**

**public Message [] getMessages() { return messages.toArray(); }**

**public void addMessage(Message m)**

**{**

**if (!messages.contains(m)) { messages.add(m);**

**m.addView(this); }**

**}**

**public void removeMessage(Message m)**

**{**

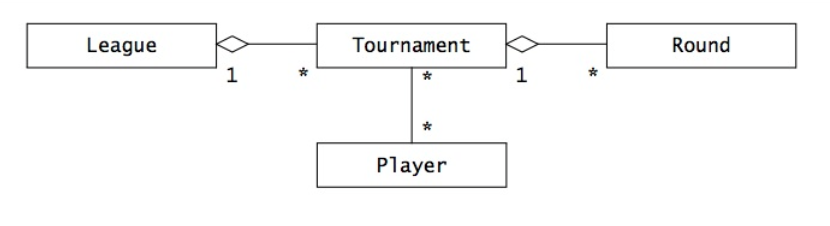
**if (messages.contains(m)) { messages.remove(m);**

**m.removeView(this); }**

**}**

**}**

1. Apply appropriate transformations to the figure below. Show class declarations including attributes and method headers. Method body may be omitted.



**The constructor of Tournament and Round set the one side of the**

**aggregation association, removing the need for a setLeague and setTournament method.**

**public class League**

**{**

**Private Set tournaments;**

**public League() { tournaments = new HashSet(); }**

**public Tournament [] getTournaments()**

**{**

**return (Tournament[])tournaments.toArray();**

**}**

**public void addTournament(Tournament t) { tournaments.add(t); }**

**public void removeTournament(Tournament t)**

**{**

**tournaments.remove(t);**

**}**

**}**

**public class Tournament**

**{**

**Private League league;**

**private Set rounds;**

**private Set players;**

**public Tournament(League l) { rounds = new HashSet();**

**players = new HashSet();**

**league = l;**

**l.addTournament(this); }**

**public League getLeague() { return league; }**

**public Round [] getRounds() { return (Round[])rounds.toArray(); }**

**public void addRound(Round r) { rounds.add(r); }**

**public void removeRound(Round r) { rounds.remove(r); }**

**public Player [] getPlayers() { return (Player[])players.toArray(); }**

**public void addPlayer(Player p)**

**{**

**if (!players.contains(p)) { players.add(p);**

**p.addTournament(this); }**

**}**

**public void removePlayer(Player p)**

**{**

**if (players.contains(p)) { players.remove(p);**

**p.removeTournament(this); }**

**}**

**}**

**public class Round**

**{**

**Private Tournament tournament;**

**public Round(Tournament t) { tournament = t;**

**t.addRound(this); }**

**public Tournament getTournament() { return tournament; }**

**}**

**public class Player**

**{**

**Private Set tournaments;**

**public Player() { tournaments = new HashSet(); }**

**public Tournament [] getTournaments()**

**{**

**return (Tournament[])tournaments.toArray();**

**}**

**public void addTournament(Tournament t)**

**{**

**if (!tournaments.contains(t)) { tournaments.add(t);**

**t.addPlayer(this); }**

**}**

**public void removeTournament(Tournament t)**

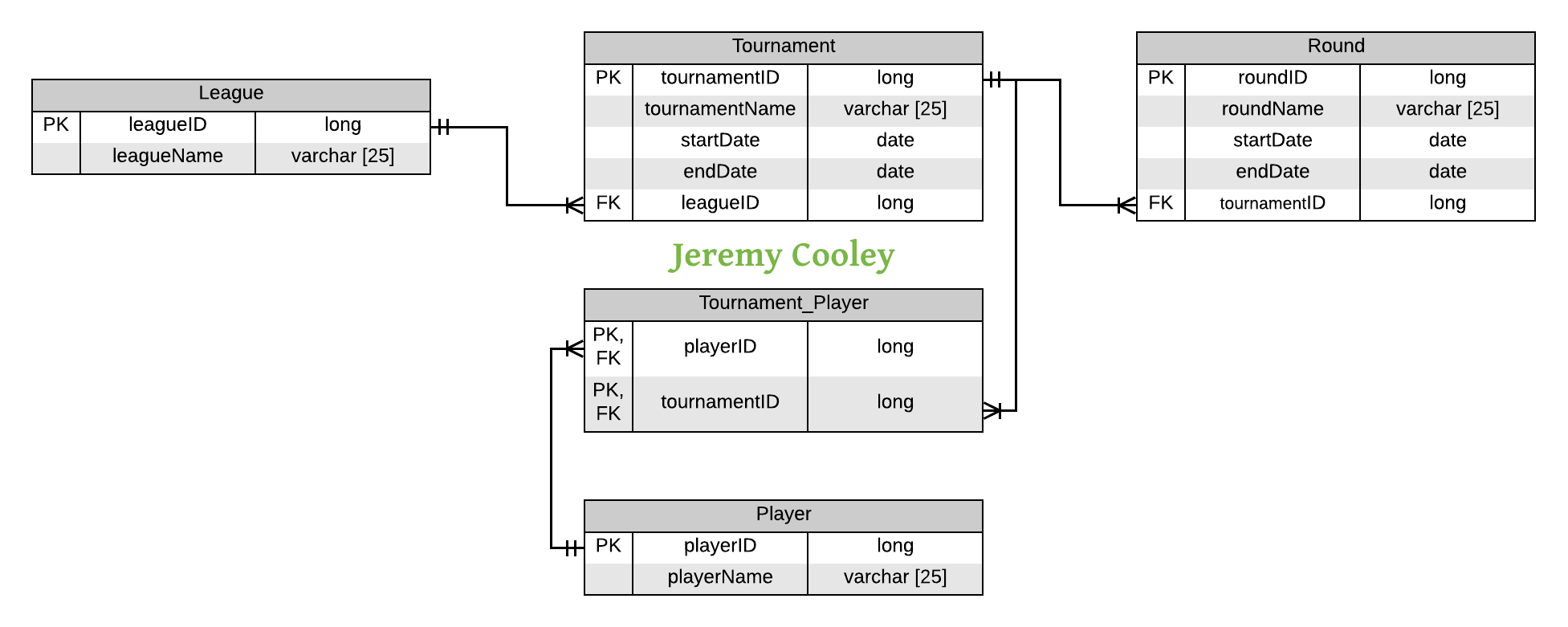
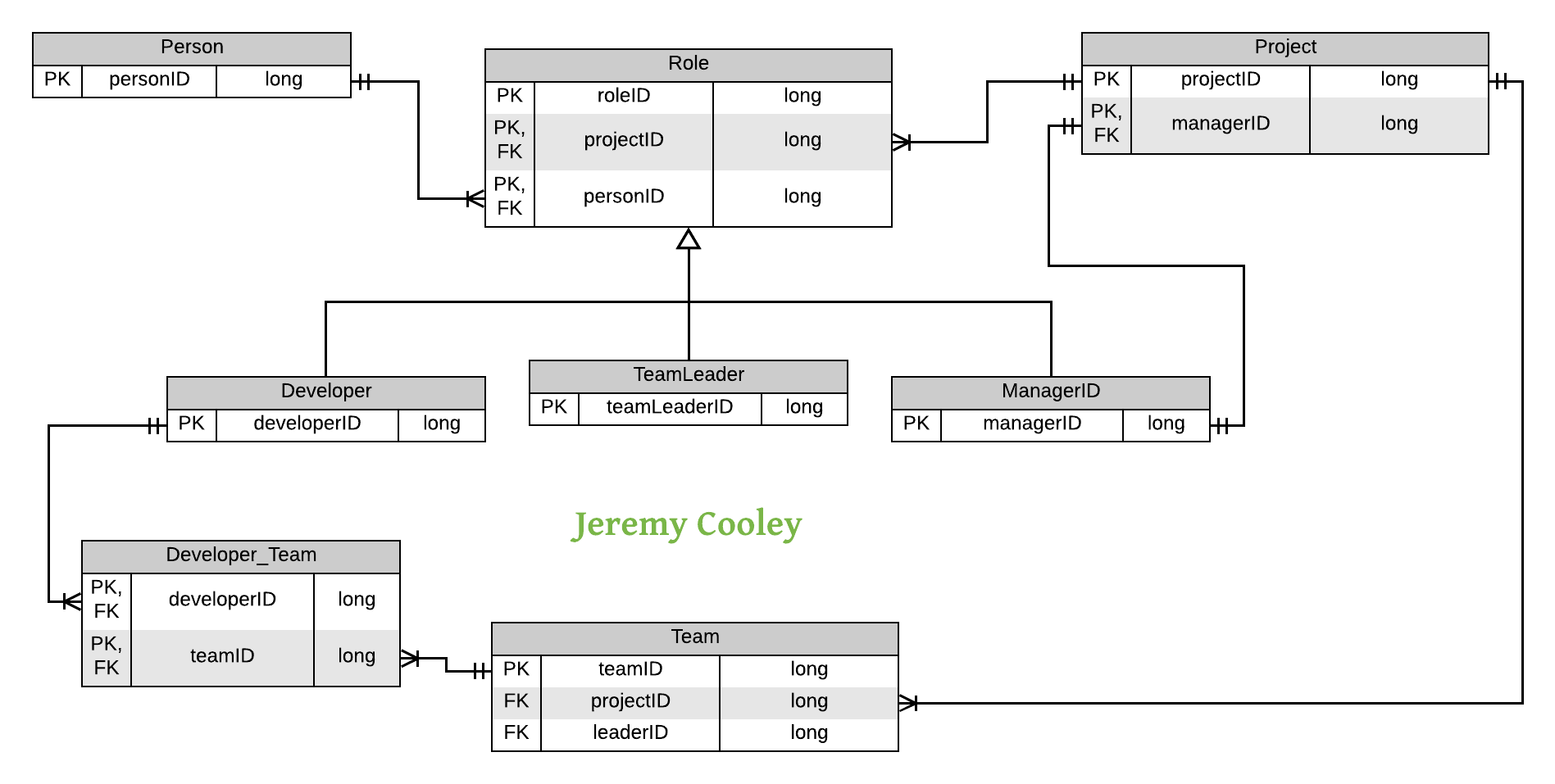
**{**

**if (tournaments.contains(t)) { tournaments.remove(t);**

**t.removePlayer(this); }**

**}**

**}**

1. Design a relational database schema for the object model in question 2 above. Assume Leagues, Tournaments, Players, and Rounds have a name attribute and a unique identifier. Additionally, Tournaments and Rounds have start and end date attributes.
2. Draw a class diagram representing the application domain facts below, and map it to a relational schema.
   1. A project involves a number of participants.
   2. Participants can take part in a project either as project manager, team leader, or League Tournament Round Player developer.
   3. Within a project, each developer and team leader is part of at least one team.
   4. A participant can take part in many projects, possibly in different roles. For example, a participant can be a developer in project A, a team leader in project B, and a project manager in project C. However, the role of a participant within a project does not change.