2.1-- REFACTORIZACIÓN del SOFTWARE

Abraham Casas-Asier Alcibar-Carlos Vilarchao

Asier:

1. **Immediately return this expression instead of assigning it to the temporary variable "events".**

```
public List<Event> getEventsAll() {
	TypedQuery<Event> query = db.createQuery("SELECT ev FROM Event ev
",Event.class);
	List<Event> events = query.getResultList();
	 return events;
}
```

Solución: El código debe simplificarse siempre que se pueda, no hace falta crear una nueva variable.

```
public List<Event> getEventsAll() {
	TypedQuery<Event> query = db.createQuery("SELECT ev FROM Event ev
",Event.class);
	return query.getResultList();
}
```

2. **Replace this if-then-else statement by a single method invocation.**

```
if (events.size() == 0)
	jButtonCreate.setEnabled(false);
else
	jButtonCreate.setEnabled(true);
```

Solución: evitar tener que hacer un if else, y comprimirlo.

```
jButtonCreate.setEnabled(events.size() != 0);
```

3. **Define a constant instead of duplicating this literal "¿Quién ganará el partido?" 3 times. (3 veces)**

```
if (Locale.getDefault().equals(new Locale("es"))) {
	q1=ev1.addQuestion("¿Quién ganará el partido?",1);
	q2=ev1.addQuestion("¿Quién meterá el primer gol?",2);
	q3=ev11.addQuestion("¿Quién ganará el partido?",1);
	q4=ev11.addQuestion("¿Cuántos goles se marcarán?",2);
	q5=ev17.addQuestion("¿Quién ganará el partido?",1);
	q6=ev17.addQuestion("¿Habrá goles en la primera parte?",2);
}else if (Locale.getDefault().equals(new Locale("en"))) {
	q1=ev1.addQuestion("Who will win the match?",1);
	q2=ev1.addQuestion("Who will score first?",2);
	q3=ev11.addQuestion("Who will win the match?",1);
	q4=ev11.addQuestion("How many goals will be scored in the match?",2);
```

```
        q5=ev17.addQuestion("Who will win the match?",1);
        q6=ev17.addQuestion("Will there be goals in the first half?",2);
}else {
        q1=ev1.addQuestion("Zeinek irabaziko du partidua?",1);
        q2=ev1.addQuestion("Zeinek sartuko du lehenengo gola?",2);
        q3=ev11.addQuestion("Zeinek irabaziko du partidua?",1);
        q4=ev11.addQuestion("Zenbat gol sartuko dira?",2);
        q5=ev17.addQuestion("Zeinek irabaziko du partidua?",1);
        q6=ev17.addQuestion("Golak sartuko dira lehenengo zatian?",2);
}
```

Solución: Debido a que al reescribir código a mano pueden suceder errores, es mejor escribirlo una sola vez y meterlo en una constante. Pero en este caso he decido dejarlo como una variable que se pueda usar en diferentes idiomas.

```
String quienGanadorPreg= "";
if (Locale.getDefault().equals(new Locale("es"))) {
        quienGanadorPreg="Â¿QuiÃ©n ganarÃ¡ el partido?";
        q1=ev1.addQuestion(quienGanadorPreg,1);
        q2=ev1.addQuestion("Â¿QuiÃ©n meterÃ¡ el primer gol?",2);
        q3=ev11.addQuestion(quienGanadorPreg,1);
        q4=ev11.addQuestion("Â¿CuÃ¡ntos goles se marcarÃ¡n?",2);
        q5=ev17.addQuestion(quienGanadorPreg,1);
        q6=ev17.addQuestion("Â¿HabrÃ¡ goles en la primera parte?",2);
}else if (Locale.getDefault().equals(new Locale("en"))) {
        quienGanadorPreg="Who will win the match?";
        q1=ev1.addQuestion(quienGanadorPreg,1);
        q2=ev1.addQuestion("Who will score first?",2);
        q3=ev11.addQuestion(quienGanadorPreg,1);
        q4=ev11.addQuestion("How many goals will be scored in the match?",2);
        q5=ev17.addQuestion(quienGanadorPreg,1);
        q6=ev17.addQuestion("Will there be goals in the first half?",2);
}else {
        quienGanadorPreg="Zeinek irabaziko du partidua?";
        q1=ev1.addQuestion(quienGanadorPreg,1);
        q2=ev1.addQuestion("Zeinek sartuko du lehenengo gola?",2);
        q3=ev11.addQuestion(quienGanadorPreg,1);
        q4=ev11.addQuestion("Zenbat gol sartuko dira?",2);
        q5=ev17.addQuestion(quienGanadorPreg,1);
        q6=ev17.addQuestion("Golak sartuko dira lehenengo zatian?",2);
}
```

4. **No he encontrado ningún método con más de 4 parámetros a parte del descrito por mi compañero Carlos Vilarchao Sáez anteriormente.**

Carlos:

1. **Refactor this method to reduce its Cognitive Complexity from 25 to the 15 allowed.**

```
public boolean ApustuaEgin (Registered u, Vector<Quote> q2, Double balioa, Integer
apustuBikoitzaGalarazi) {

        Registered user = (Registered) db.find(Registered.class, u.getUsername());
        Boolean b;

if(user.getDirukop()>=balioa) {
            db.getTransaction().begin();
            ApustuAnitza apustuAnitza = new ApustuAnitza(user, balioa);
            db.persist(apustuAnitza);

    for(Quote quo: q2) {
        Quote kuote = db.find(Quote.class, quo.getQuoteNumber());
        Apustua ap = new Apustua(apustuAnitza, kuote);
        db.persist(ap);
        apustuAnitza.addApustua(ap);
        kuote.addApustua(ap);
    }
    db.getTransaction().commit();
    db.getTransaction().begin();

    if(apustuBikoitzaGalarazi==-1) {

        apustuBikoitzaGalarazi=apustuAnitza.getApustuAnitzaNumber();
    }
    apustuAnitza.setApustuKopia(apustuBikoitzaGalarazi);
    user.updateDiruKontua(-balioa);
    Transaction t = new Transaction(user, balioa, new Date(), ae);
    user.addApustuAnitza(apustuAnitza);

    for(Apustua a: apustuAnitza.getApustuak()) {
        Apustua apu = db.find(Apustua.class, a.getApostuaNumber());
        Quote q = db.find(Quote.class, apu.getKuota().getQuoteNumber());
        Sport spo =q.getQuestion().getEvent().getSport();
        spo.setApustuKantitatea(spo.getApustuKantitatea()+1);
    }
    user.addTransaction(t);
    db.persist(t);
    db.getTransaction().commit();

    for(Jarraitzailea reg:user.getJarraitzaileLista()) {
        Jarraitzailea erab=db.find(Jarraitzailea.class, reg.getJarraitzaileaNumber());
        b=true;
        for(ApustuAnitza apu: erab.getNork().getApustuAnitzak()) {
```

```
                    if(apu.getApustuKopia().equals(apustuAnitza.getApustuKopia())) {
                            b=false;
                    }
            }
            if(b) {
                    if(erab.getNork().getDiruLimitea()<balioa) {
                            this.ApustuaEgin(erab.getNork(), q2, erab.getNork().getDiruLimitea(),
                    apustuBikoitzaGalarazi);
                    }
            else{
                    this.ApustuaEgin(erab.getNork(), q2, balioa, apustuBikoitzaGalarazi);
            }
            }
            }
            return true;
            }
else{
            return false;
}
}
```

Solución: Simplificamos la complejidad cognitiva del método de 25 a 15, extrayendo métodos del original a nuevos métodos, los que sacamos son los que añaden complejidad al método, los fors.

```
public boolean ApustuaEgin(Registered u, Vector<Quote> q2, Double balioa, Integer
apustuBikoitzaGalarazi) {
        Registered user = (Registered) db.find(Registered.class, u.getUsername());
        Boolean b;
        if(user.getDirukop()>=balioa) {
                db.getTransaction().begin();
                ApustuAnitza apustuAnitza = new ApustuAnitza(user, balioa);
                db.persist(apustuAnitza);
                forListaQuotes(q2, apustuAnitza);
                db.getTransaction().commit();
                db.getTransaction().begin();
                if(apustuBikoitzaGalarazi==-1) {
                        apustuBikoitzaGalarazi=apustuAnitza.getApustuAnitzaNumber();
                }
                apustuAnitza.setApustuKopia(apustuBikoitzaGalarazi);
                user.updateDiruKontua(-balioa);
                Transaction t = new Transaction(user, balioa, new Date(), ae);
                user.addApustuAnitza(apustuAnitza);
                forListaApuestas(apustuAnitza);
                user.addTransaction(t);
                db.persist(t);
                db.getTransaction().commit();
                forJarraitzaileak(q2, balioa, apustuBikoitzaGalarazi, user, apustuAnitza);
```

```
                return true;
        }else{
                return false;
        }

    }

    private void forJarraitzaileak(Vector<Quote> q2, Double balioa, Integer
apustuBikoitzaGalarazi, Registered user,
                ApustuAnitza apustuAnitza) {
        Boolean b;
        for(Jarraitzailea reg:user.getJarraitzaileLista()) {
                Jarraitzailea erab=db.find(Jarraitzailea.class, reg.getJarraitzaileaNumber());
                b=true;
                for(ApustuAnitza apu: erab.getNork().getApustuAnitzak()) {
                        if(apu.getApustuKopia().equals(apustuAnitza.getApustuKopia())) {
                                b=false;
                        }
                }
                if(b) {
                        if(erab.getNork().getDiruLimitea()<balioa) {
                                this.ApustuaEgin(erab.getNork(), q2,
erab.getNork().getDiruLimitea(), apustuBikoitzaGalarazi);
                        }else{
                                this.ApustuaEgin(erab.getNork(), q2, balioa,
apustuBikoitzaGalarazi);
                        }
                }
        }
    }

    private void forListaApuestas(ApustuAnitza apustuAnitza) {
        for(Apustua a: apustuAnitza.getApustuak()) {
                Apustua apu = db.find(Apustua.class, a.getApostuaNumber());
                Quote q = db.find(Quote.class, apu.getKuota().getQuoteNumber());
                Sport spo =q.getQuestion().getEvent().getSport();
                spo.setApustuKantitatea(spo.getApustuKantitatea()+1);

        }
    }

    private void forListaQuotes(Vector<Quote> q2, ApustuAnitza apustuAnitza) {
        for(Quote quo: q2) {
                Quote kuote = db.find(Quote.class, quo.getQuoteNumber());
                Apustua ap = new Apustua(apustuAnitza, kuote);
                db.persist(ap);
                apustuAnitza.addApustua(ap);
                kuote.addApustua(ap);
```

```
        }
    }
```

2. **Una forma habitual de evaluar objetivamente la complejidad es contar el número de caminos posibles a través de un fragmento de código**

```
public boolean ApustuaEgin (Registered u, Vector<Quote> q2, Double balioa, Integer
apustuBikoitzaGalarazi) {

            Registered user = (Registered) db.find(Registered.class, u.getUsername());
            Boolean b;

if(user.getDirukop()>=balioa) {
                db.getTransaction().begin();
                ApustuAnitza apustuAnitza = new ApustuAnitza(user, balioa);
                db.persist(apustuAnitza);

    for(Quote quo: q2) {
            Quote kuote = db.find(Quote.class, quo.getQuoteNumber());
            Apustua ap = new Apustua(apustuAnitza, kuote);
            db.persist(ap);
            apustuAnitza.addApustua(ap);
            kuote.addApustua(ap);
    }
    db.getTransaction().commit();
    db.getTransaction().begin();

    if(apustuBikoitzaGalarazi==-1) {

            apustuBikoitzaGalarazi=apustuAnitza.getApustuAnitzaNumber();
    }
    apustuAnitza.setApustuKopia(apustuBikoitzaGalarazi);
    user.updateDiruKontua(-balioa);
    Transaction t = new Transaction(user, balioa, new Date(), ae);
    user.addApustuAnitza(apustuAnitza);

    for(Apustua a: apustuAnitza.getApustuak()) {
            Apustua apu = db.find(Apustua.class, a.getApostuaNumber());
            Quote q = db.find(Quote.class, apu.getKuota().getQuoteNumber());
            Sport spo =q.getQuestion().getEvent().getSport();
            spo.setApustuKantitatea(spo.getApustuKantitatea()+1);
    }
    user.addTransaction(t);
    db.persist(t);
    db.getTransaction().commit();

    for(Jarraitzailea reg:user.getJarraitzaileLista()) {
```

```
                Jarraitzailea erab=db.find(Jarraitzailea.class, reg.getJarraitzaileaNumber());
                b=true;
                for(ApustuAnitza apu: erab.getNork().getApustuAnitzak()) {
                        if(apu.getApustuKopia().equals(apustuAnitza.getApustuKopia())) {
                                b=false;
                }
        }
        if(b) {
                if(erab.getNork().getDiruLimitea()<balioa) {
                        this.ApustuaEgin(erab.getNork(), q2, erab.getNork().getDiruLimitea(),
                apustuBikoitzaGalarazi);
                }
        else{
                this.ApustuaEgin(erab.getNork(), q2, balioa, apustuBikoitzaGalarazi);
        }
        }
        }
        return true;
        }
else{
        return false;
}
}
```

Solución: el método anterior tiene una complejidad ciclomática de 10 debido a su gran cantidad de bucles y de condiciones, por tanto tenía gran cantidad de rutas que podía tomar el método, pero al hacer extracción de métodos en el punto anterior esta complejidad se simplifica a 3 rutas.

```
public boolean ApustuaEgin(Registered u, Vector<Quote> q2, Double balioa, Integer
apustuBikoitzaGalarazi) {
        Registered user = (Registered) db.find(Registered.class, u.getUsername());
        Boolean b;
        if(user.getDirukop()>=balioa) {
                db.getTransaction().begin();
                ApustuAnitza apustuAnitza = new ApustuAnitza(user, balioa);
                db.persist(apustuAnitza);
                forListaQuotes(q2, apustuAnitza);
                db.getTransaction().commit();
                db.getTransaction().begin();
                if(apustuBikoitzaGalarazi==-1) {
                        apustuBikoitzaGalarazi=apustuAnitza.getApustuAnitzaNumber();
                }
                apustuAnitza.setApustuKopia(apustuBikoitzaGalarazi);
                user.updateDiruKontua(-balioa);
                Transaction t = new Transaction(user, balioa, new Date(), ae);
                user.addApustuAnitza(apustuAnitza);
                forListaApuestas(apustuAnitza);
```

```
                user.addTransaction(t);
                db.persist(t);
                db.getTransaction().commit();
                forJarraitzaileak(q2, balioa, apustuBikoitzaGalarazi, user, apustuAnitza);
                return true;
        }else{
                return false;
        }

  }
```

3. **Define a constant instead of duplicating this literal "ApustuaEgin" 12 times.**

Transaction t1 = new Transaction(reg1, apA1.getBalioa(), new Date(), "ApustuaEgin");
Transaction t3 = new Transaction(reg2, apA4.getBalioa(), new Date(), "ApustuaEgin");
Transaction t4 = new Transaction(reg3, apA5.getBalioa(), new Date(), "ApustuaEgin");
Transaction t5 = new Transaction(reg4, apA3.getBalioa(), new Date(), "ApustuaEgin");
Transaction t6 = new Transaction(reg4, apA6.getBalioa(), new Date(), "ApustuaEgin");
Transaction t7 = new Transaction(reg1, apA7.getBalioa(), new Date(), "ApustuaEgin");
Transaction t8 = new Transaction(reg1, apA8.getBalioa(), new Date(), "ApustuaEgin");
Transaction t9 = new Transaction(reg2, apA9.getBalioa(), new Date(), "ApustuaEgin");
Transaction t10 = new Transaction(reg2, apA10.getBalioa(), new Date(), "ApustuaEgin");
Transaction t11 = new Transaction(reg3, apA11.getBalioa(), new Date(), "ApustuaEgin");
Transaction t12 = new Transaction(reg3, apA12.getBalioa(), new Date(), "ApustuaEgin");

Solución: Debido a que al reescribir código a mano pueden suceder errores, es mejor escribirlo una sola vez y meterlo en una constante, además nos ahorra tiempo.

```
private static final String ae = "ApustuaEgin";

Transaction t1 = new Transaction(reg1, apA1.getBalioa(), new Date(), ae);
Transaction t3 = new Transaction(reg2, apA4.getBalioa(), new Date(), ae);
Transaction t4 = new Transaction(reg3, apA5.getBalioa(), new Date(), ae);
Transaction t5 = new Transaction(reg4, apA3.getBalioa(), new Date(), ae);
Transaction t6 = new Transaction(reg4, apA6.getBalioa(), new Date(), ae);
Transaction t7 = new Transaction(reg1, apA7.getBalioa(), new Date(), ae);
Transaction t8 = new Transaction(reg1, apA8.getBalioa(), new Date(), ae);
Transaction t9 = new Transaction(reg2, apA9.getBalioa(), new Date(), ae);
Transaction t10 = new Transaction(reg2, apA10.getBalioa(), new Date(), ae);
Transaction t11 = new Transaction(reg3, apA11.getBalioa(), new Date(), ae);
Transaction t12 = new Transaction(reg3, apA12.getBalioa(), new Date(), ae);
```

4. **Método con más de 4 parámetros. Justo este metodo con mas de 4 parámetros es uno que hemos acabado extrayendo en el punto 1 que es:**

   private void forJarraitzaileak(Vector<Quote> q2, Double balioa, Integer apustuBikoitzaGalarazi, Registered user, ApustuAnitza apustuAnitza) {

```
            Boolean b;
            for(Jarraitzailea reg:user.getJarraitzaileLista()) {
                  Jarraitzailea erab=db.find(Jarraitzailea.class,
      reg.getJarraitzaileaNumber());
                  b=true;
                  for(ApustuAnitza apu: erab.getNork().getApustuAnitzak()) {

      if(apu.getApustuKopia().equals(apustuAnitza.getApustuKopia())) {
                              b=false;
                        }
                  }
                  if(b) {
                        if(erab.getNork().getDiruLimitea()<balioa) {
                              this.ApustuaEgin(erab.getNork(), q2,
      erab.getNork().getDiruLimitea(), apustuBikoitzaGalarazi);
                        }else{
                              this.ApustuaEgin(erab.getNork(), q2, balioa,
      apustuBikoitzaGalarazi);
                        }
                  }
            }
        }
```

Solución: la solución sería convertir uno de los parámetros a un objeto , esto mejorará la capacidad de mantenimiento porque mantener el número de parámetros bajo hace que las unidades sean más fáciles de entender y reutilizar. El problema es que no entiendo como se realizaría esta corrección sin estropear todo el código de ambos métodos. He estado buscando otros parecidos con más de 4 parámetros pero no consigo encontrar o no es posible quitarlos sin estropear todo el código.

Abraham:

1. **Immediately return this expression instead of assigning it to the temporary variable "teams".**

```
public List<Team> getAllTeams() {
      TypedQuery<Team> query = db.createQuery("SELECT t FROM Team t
",Team.class);
      List<Team> teams = query.getResultList();
      return teams;
}
```

Solución: El código debe simplificarse siempre que se pueda, no hace falta crear una nueva variable.

```
public List<Team> getAllTeams() {
      TypedQuery<Team> query = db.createQuery("SELECT t FROM Team t
",Team.class);
```

```
        return query.getResultList();
}
```

2. **Una forma habitual de evaluar objetivamente la complejidad es contar el número de caminos posibles a través de un fragmento de código:**

```
        public void EmaitzakIpini(Quote quote) throws EventNotFinished{

                Quote q = db.find(Quote.class, quote);
                String result = q.getForecast();

                if(new
        Date().compareTo(q.getQuestion().getEvent().getEventDate())<0)
                        throw new EventNotFinished();

                Vector<Apustua> listApustuak = q.getApustuak();
                db.getTransaction().begin();
                Question que = q.getQuestion();
                Question question = db.find(Question.class, que);
                question.setResult(result);
                for(Quote quo: question.getQuotes()) {
                        for(Apustua apu: quo.getApustuak()) {

                                Boolean b=apu.galdutaMarkatu(quo);
                                if(b) {
                                        apu.getApustuAnitza().setEgoera("galduta");
                                }else {
                                        apu.setEgoera("irabazita");
                                }
                        }
                }
                db.getTransaction().commit();
                for(Apustua a : listApustuak) {
                        db.getTransaction().begin();
                        Boolean bool=a.getApustuAnitza().irabazitaMarkatu();
                        db.getTransaction().commit();
                        if(bool) {
                                this.ApustuaIrabazi(a.getApustuAnitza());
                        }
                }
        }
```

Solución: El método tiene una complejidad ciclomática mayor que cuatro, para reducir dicha complejidad una de las soluciones posibles es dividir el método en métodos para que cada uno tenga su propia responsabilidad única:

```
public void EmaitzakIpini(Quote quote) throws EventNotFinished {
   Quote q = db.find(Quote.class, quote);
   String result = q.getForecast();
```

```
    if (eventNotFinished(q)) {
        throw new EventNotFinished();
    }

    processQuotesAndApustuak(q, result);
}

private boolean eventNotFinished(Quote q) {
    Date currentDate = new Date();
    Date eventDate = q.getQuestion().getEvent().getEventDate();
    return currentDate.compareTo(eventDate) < 0;
}

private void processQuotesAndApustuak(Quote q, String result) {
    db.getTransaction().begin();
    Question question = db.find(Question.class, q.getQuestion());

    question.setResult(result);

    for (Quote quo : question.getQuotes()) {
        for (Apustua apu : quo.getApustuak()) {
            boolean isLost = apu.galdutaMarkatu(quo);

            if (isLost) {
                apu.getApustuAnitza().setEgoera("galduta");
            } else {
                apu.setEgoera("irabazita");
            }
        }
    }

    db.getTransaction().commit();

    for (Apustua a : q.getApustuak()) {
        db.getTransaction().begin();
        boolean isWon = a.getApustuAnitza().irabazitaMarkatu();
        db.getTransaction().commit();

        if (isWon) {
            this.ApustuaIrabazi(a.getApustuAnitza());
        }
    }
}
```

3. **Define a constant instead of duplicating this literal "DiruaSartu" 4 times.**

this.DiruaSartu(reg1, 50.0, new Date(), "DiruaSartu");

this.DiruaSartu(reg2, 50.0, new Date(), "DiruaSartu");
this.DiruaSartu(reg3, 50.0, new Date(), "DiruaSartu");
this.DiruaSartu(reg4, 50.0, new Date(), "DiruaSartu");

Solución: Como el anterior.

```
private static final String DIRUA_SARTU = "DiruaSartu";

this.DiruaSartu(reg1, 50.0, new Date(), DIRUA_SARTU);
this.DiruaSartu(reg2, 50.0, new Date(), DIRUA_SARTU);
this.DiruaSartu(reg3, 50.0, new Date(), DIRUA_SARTU);
this.DiruaSartu(reg4, 50.0, new Date(), DIRUA_SARTU);
```

4. **No he encontrado ningún método con más de 4 parámetros a parte del descrito por mi compañero Carlos Vilarchao Sáez anteriormente.**