Programación 4

EXAMEN FEBRERO 2014

SOLUCIÓN

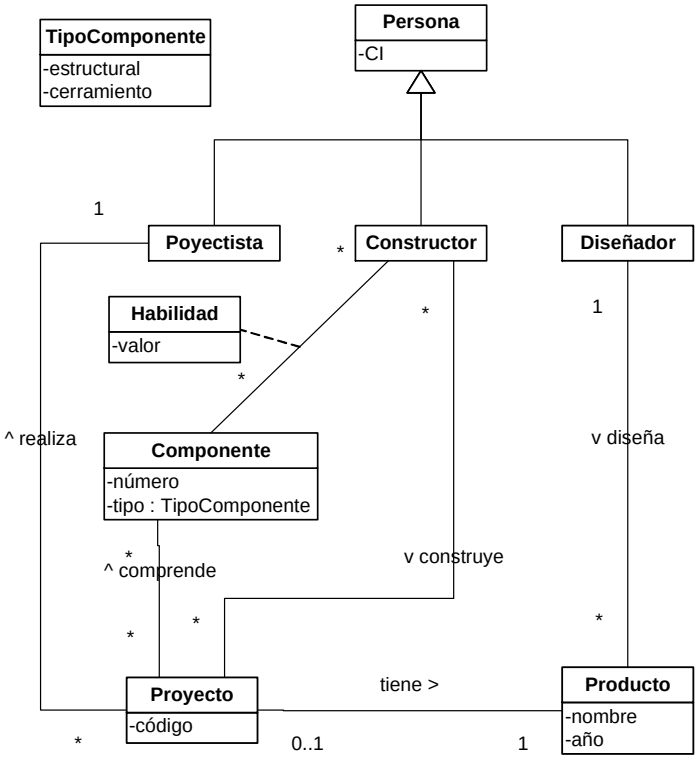
Problema 1 (30 puntos)

a.

- *Evento del sistema*: estímulo externo, generado por un actor ante el cual el sistema debe reaccionar.
- *Operación del sistema*: define una interacción entre un actor y el sistema, es disparada por un evento del sistema y es ejecutada por la instancia que representa al sistema, en respuesta a dicho evento.

b.

i.



ii.

- Una Persona es identificada por su CI.
- Un Componente es identificado por su número.
- Un Proyecto es identificado por su código.
- Un Producto es identificado por su nombre.

- El valor de una Habilidad pertenece al rango 0..5.

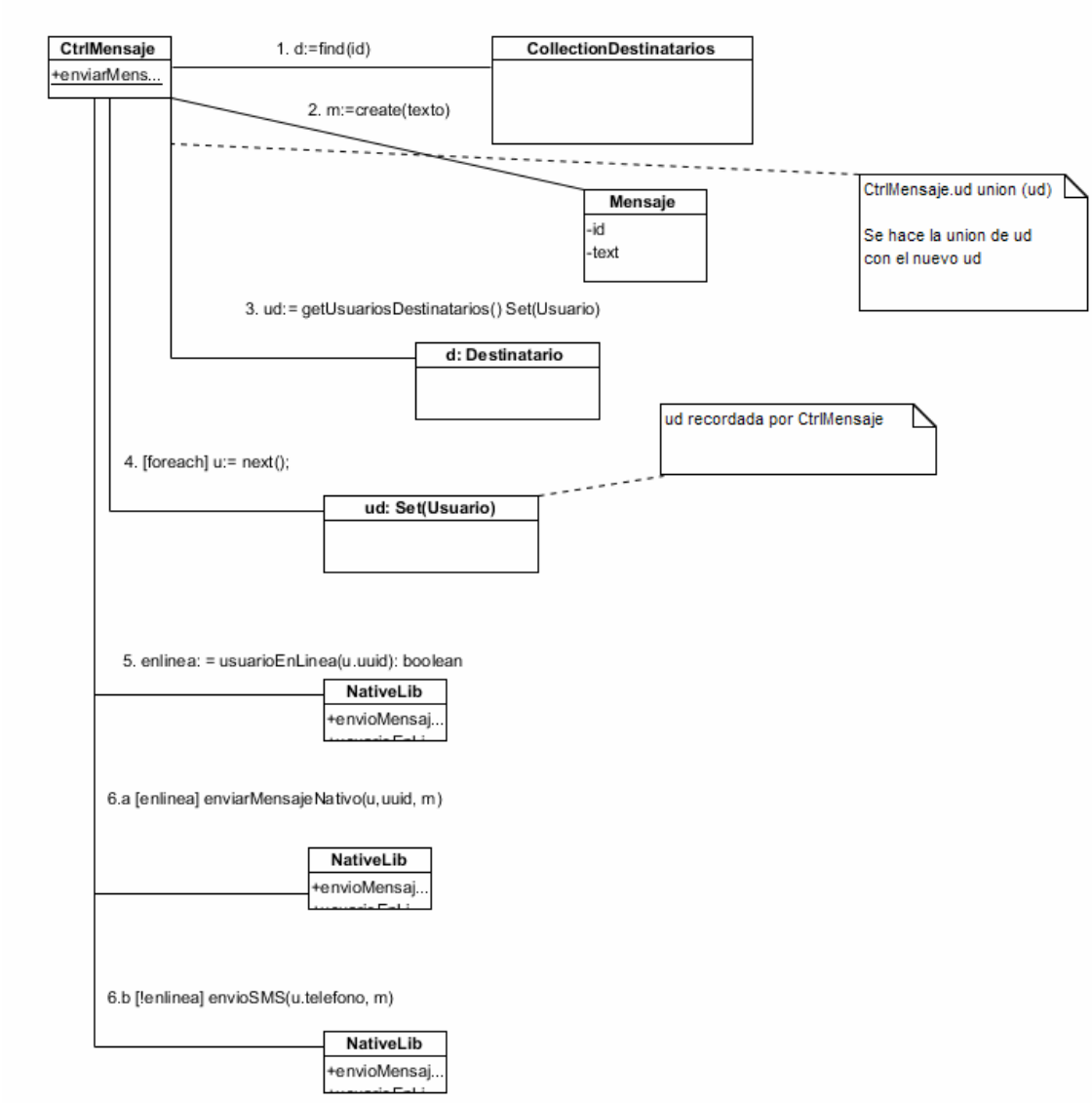
- Un Constructor puede construir un Proyecto si tiene valor de Habilidad mayor o igual a 2 en todos los Componentes del mismo.

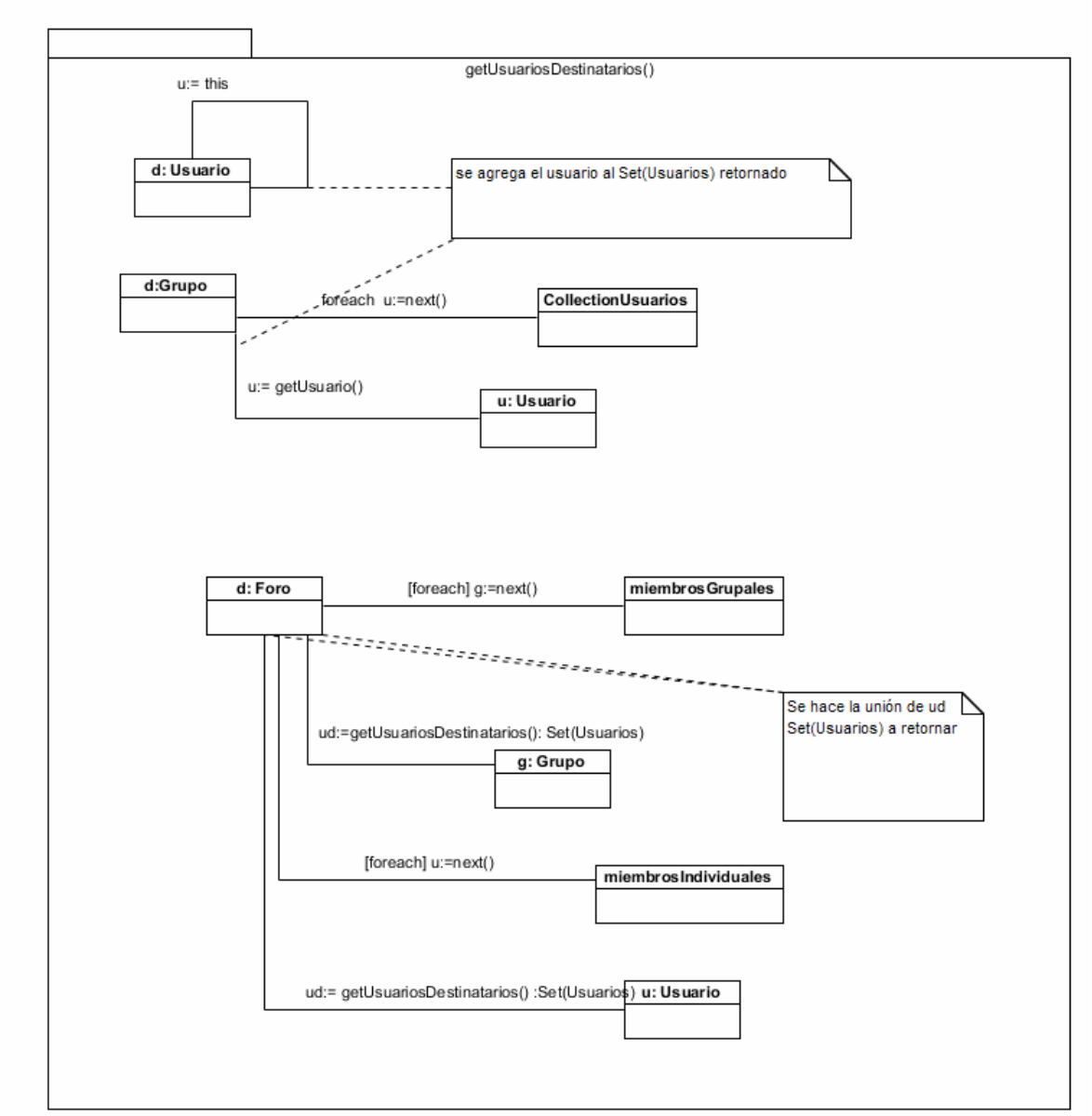
Problema 2 (35 puntos)

a. Expert, Creator, Bajo Acoplamiento, Alta Cohesión, No Hables con Extraños y Controller.

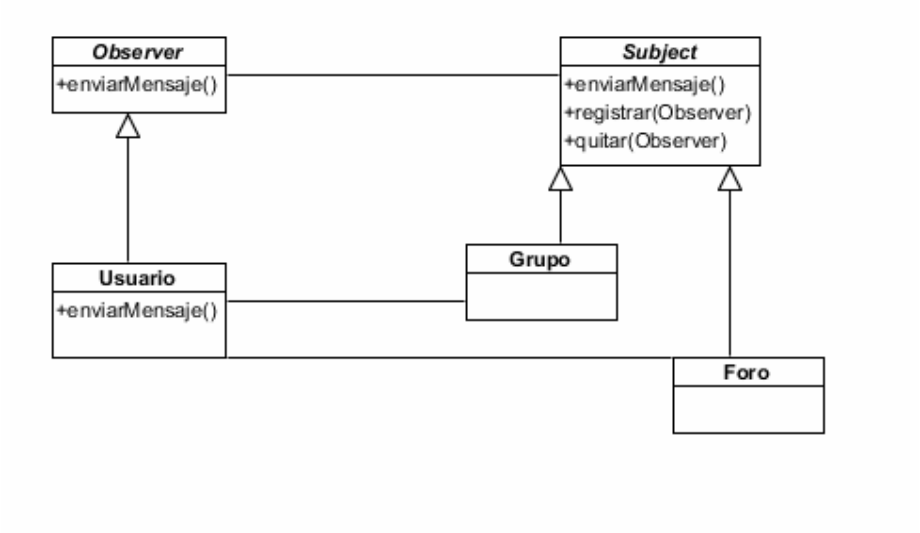
b.

i.




getUsuariosDestinatarios()
u:= this
d: Usuario
se agrega el usuario al Set(Usuarios) retornado
d:Grupo
foreach u:=next()
CollectionUsuarios
u:= getUsuario()
u: Usuario
d: Foro
[foreach] g:=next()
miembrosGrupales
Se hace la unión de ud
Set(Usuarios) a retornar
ud:=getUsuariosDestinatarios(): Set(Usuarios)
g: Grupo
[foreach] u:=next()
miembrosIndividuales
ud:= getUsuariosDestinatarios() :Set(Usuarios)
u: Usuario

ii



Se utiliza el patrón Observer, con los siguientes roles:

- Concrete Observer: Usuario
- Concrete Subject: Grupo y Foro

Problema 3 (35 puntos)

a.

```
// PackageManager.h

class PackageManager{
  private:
    IDictionary* packages;

  public:
    PackageManager(IDictionary*);
    void instalar(String);
    void actualizar(String);
    virtual ~PackageManager();
};

// PackageManager.cpp

PackageManager::PackageManager(IDictionary* packages){
  this->packages = packages;
}

void PackageManager::instalar(String nombre){
  IKey* ks = new KeyString(nombre);
  Package* p = (Package*) packages->find(ks);
  delete ks;
  if (!p->isInstalled())
    return;

  ICollection* pkgs = p->paquetesRequeridos();
  pkgs->add(p);
```

```
  IIterator* it = pkgs->getIterator();
  for(it; it->hasNext(); it->next()){
    Package* pkg = (Package*) it->current();

    String n = pkg->getNombre();
    DataPackageVersion* dpv = Repository::descargar(n);

    pkg->addVersion(dpv);

    delete dpv;
  }
  delete it;
  delete pkgs;
}

void PackageManager::actualizar(String nombre){
  IKey* ks = new KeyString(nombre);
  Package* p = (Package*) packages->find(ks);
  delete ks;

  String n = p->getNombre();

  int actual = p->obtenerVersion();

  int ultima = Repository::ultimaVersion(nombre);

  if (ultima > actual){
    DataPackageVersion* dpv = Repository::descargar(n);
    p->addVersion(dpv);

    delete dpv;
  }
}

PackageManager::~PackageManager(){
  IIterator* elems = packages->getElemIterator();
  IIterator* keys = packages->getKeyIterator();

  for(elems;elems->hasNext();elems->next()){
    Package* current = (Package*) it->current();
    delete elems->current;
  }

  for(keys;keys->hasNext();keys->next()){
    delete keys->current;
  }

  delete elems;
  delete keys;
  delete packages;
}

// Package.h

class Package : public ICollectible{
  private:
    ICollection* depende_de;
    String nombre;
    String autor;
    Version* current;
    ICollection* versiones;
```

```
  public:
    Package(ICollection*,String, String);
    ICollection* paquetesRequeridos();
    void addVersion(DataPackageVersion*);
    int obtenerVersion();
    bool isInstalled();
    virtual ~Package();
};

// Package.cpp

Package::Package(ICollection*  dependencias,  String  nombre,  String
autor){
  this->depende_de = dependencias;
  this->nombre = nombre;
  this->autor = autor;
  this->versiones = new Lista();
}

ICollection* Package::paquetesRequeridos(){
  IIterator* it = depende_de->getIterator();
  ICollection* result = new Lista();
  for(it;it->hasNext();it->next()){
    Package* p = (Package*) it->current();

    if(!p->isInstalled()){

      ICollection* dep = p->paquetesRequeridos();
      IIterator* it_dep = dep->getIterator();
      for(it_dep;it_dep->hasNext();it_dep->next()){
        result->add(it_dep->current);
      }
      delete
      delete it_dep;

      result->add(p);
    }
  }
  delete it;
  return result;
}

void Package::addVersion(DataPackageVersion* dpv){
  Version* v = new Version(dpv->getVersion(), dpv->getRuta());
  this->versiones->add(v);
  this->current = v;
}

int Package::obtenerVersion(){
  return current->getVersion();
}

bool Package::isInstalled(){
  return !(versiones->isEmpty());
}

Package::~Package(){
  delete depende_de;

  IIterator* it = versiones->getIterator();
```

```
  for(it;it->hasNext();it->next()){
    Version* v = (Version*) it->current();
    delete v;
  }
  delete it;
  delete versiones;
}
```

b.

```
// AbstractPackageManager.h

class AbstractPackageManager{
  public:
    virtual void instalar(String)=0;
    virtual void actualizar(String)=0;
    virtual ~AbstractPackageManager();
};

// AbstractPackageManager.cpp

AbstractPackageManager::~AbstractPackageManager{}

// MultiEnvironmentPackageManager.h

class MultiEnvironmentPackageManager : public AbstractPackageManager{

  private:
    IDictionary* environments;
    PackageManager* currentEnvironment;

  public:
    MultiEnvironmentPackageManager();
    void instalar(String);
    void actualizar(String);
    void addEnvironment(PackageManager*);
    void setCurrentEnvironment(String);
    virtual ~MultiEnvironmentPackageManager();
};

// MultiEnvironmentPackageManager.cpp

MultiEnvironmentPackageManager::MultiEnvironmentPackageManager(){
  environments = new Diccionario();
}

void MultiEnvironmentPackageManager::instalar(String nombre){
  currentEnvironment->instalar(nombre);
}

void MultiEnvironmentPackageManager::actualizar(String nombre){
  currentEnvironment->actualizar(nombre);
}

void     MultiEnvironmentPackageManager::addEnvironment(PackageManager*
pm){
  String nombre = pm->getNombre();
  environments->add(new KeyString(nombre), pm);
}
```

```
void       MultiEnvironmentPackageManager::setCurrentEnvironment(String
nombre){
  IKey* ks = new KeyString(nombre);
  this->current = (PackageManager*) environments->find(ks);
  delete ks;
}

MultiEnvironmentPackageManager::~MultiEnvironmentPackageManager(){
  IIterator* it = environments->getIterator();
  for(it;it->hasNext();it->next()){
    PackageManager* pm = (PackageManager*) it->current();
    delete pm;
  }
  delete it;
  delete environments;
}
```