Programación 4

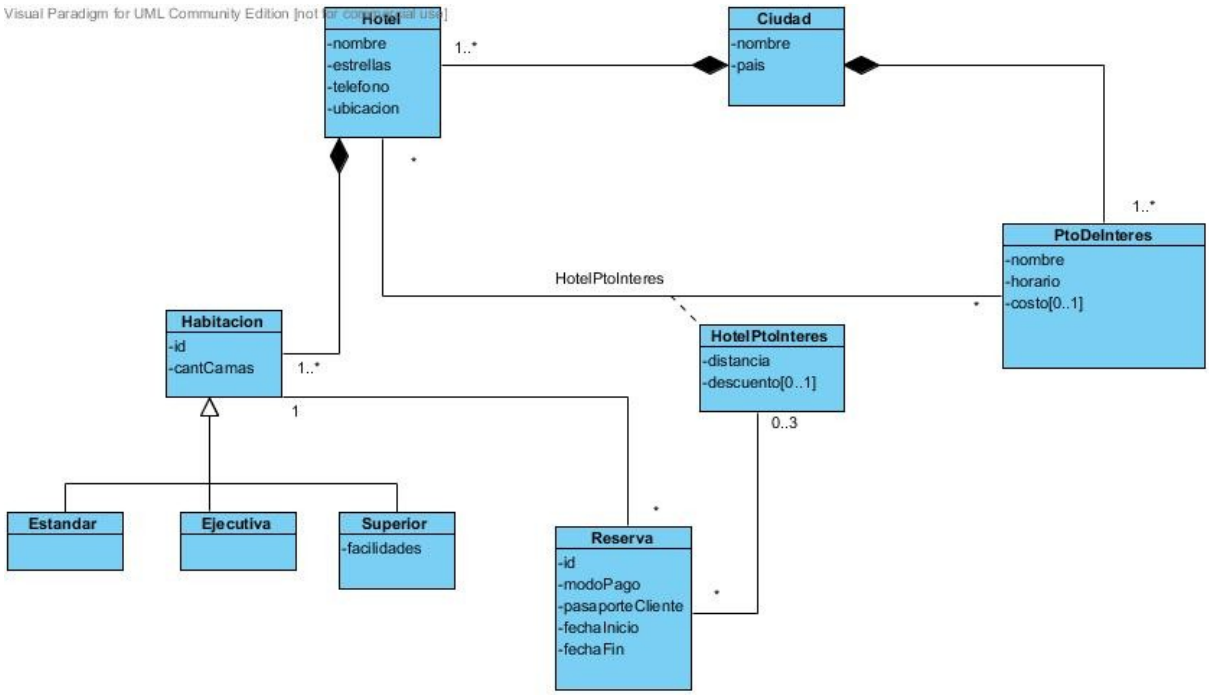
EXAMEN JULIO/AGOSTO 2013

SOLUCIÓN

Problema 1 (30 puntos)

(a) Describe una asociación que refiere a una familia de relaciones entre los objetos sobre las que se perciben propiedades que son propias de las relaciones.
Dicho de otra forma, un tipo asociativo es un elemento del modelo de dominio que es tanto clase (concepto) como asociación. Un tipo asociativo es usado para permitir agregar propiedades a una asociación.
Un ejemplo de uso: empresa que contratan personas donde el sueldo depende de la persona y de la empresa. El sueldo se representa como un atributo del tipo asociativo de la asociacion entre empresa y persona.

(b)



Restricciones no estructurales:

- [El nombre de Hotel es único]
- [El nombre de Ciudad es único]
- [El nombre de PtodeInteres es único por ciudad]
- El id de Habitación es único por Hotel
- El id de Reserva es único
- Descuento debe tener un valor entre 0 y 100, distancia > 0, cantCamas > 0, estrellas > 0.
- La fecha de inicio de una reserva debe ser menor a la fechaFin.
- Si existe descuento en un link (h, p) entonces existe costo en p
- Si existe link entre Reserva y HotelPtoInteres hp entonces existe descuento en hp.

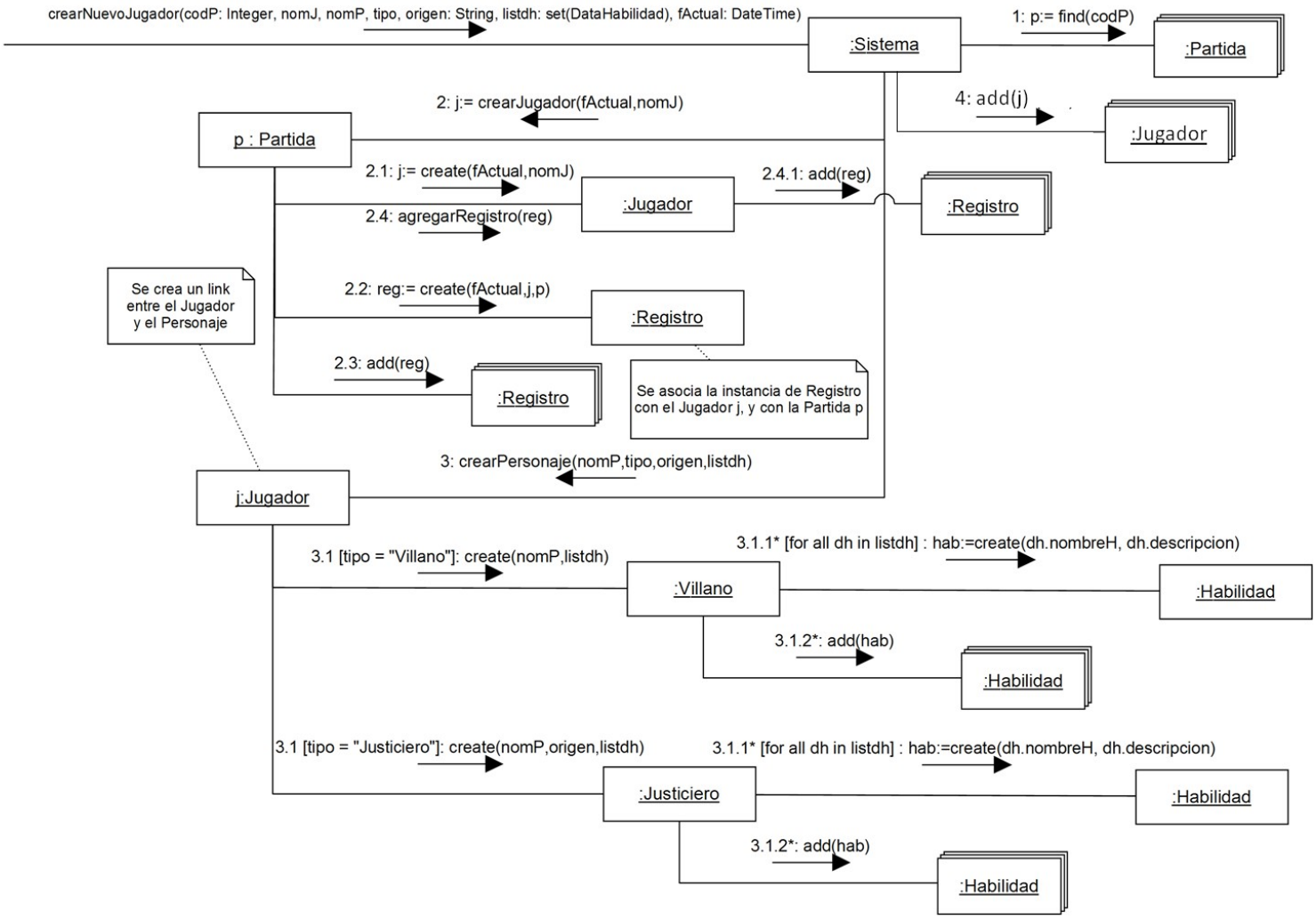
- No existen Reservas para una misma Habitación cuyas fechas de estadia se superpongan.
- El HotelPtoInteres de interés asociada a una reserva debe corresponder a una reserva para la habitación del mismo hotel.
- Si (h, p) es un link de HotelPtoInteres, entonces el Hotel h y el Punto de Interes p están en la misma ciudad.

Problema 2 (35 puntos)

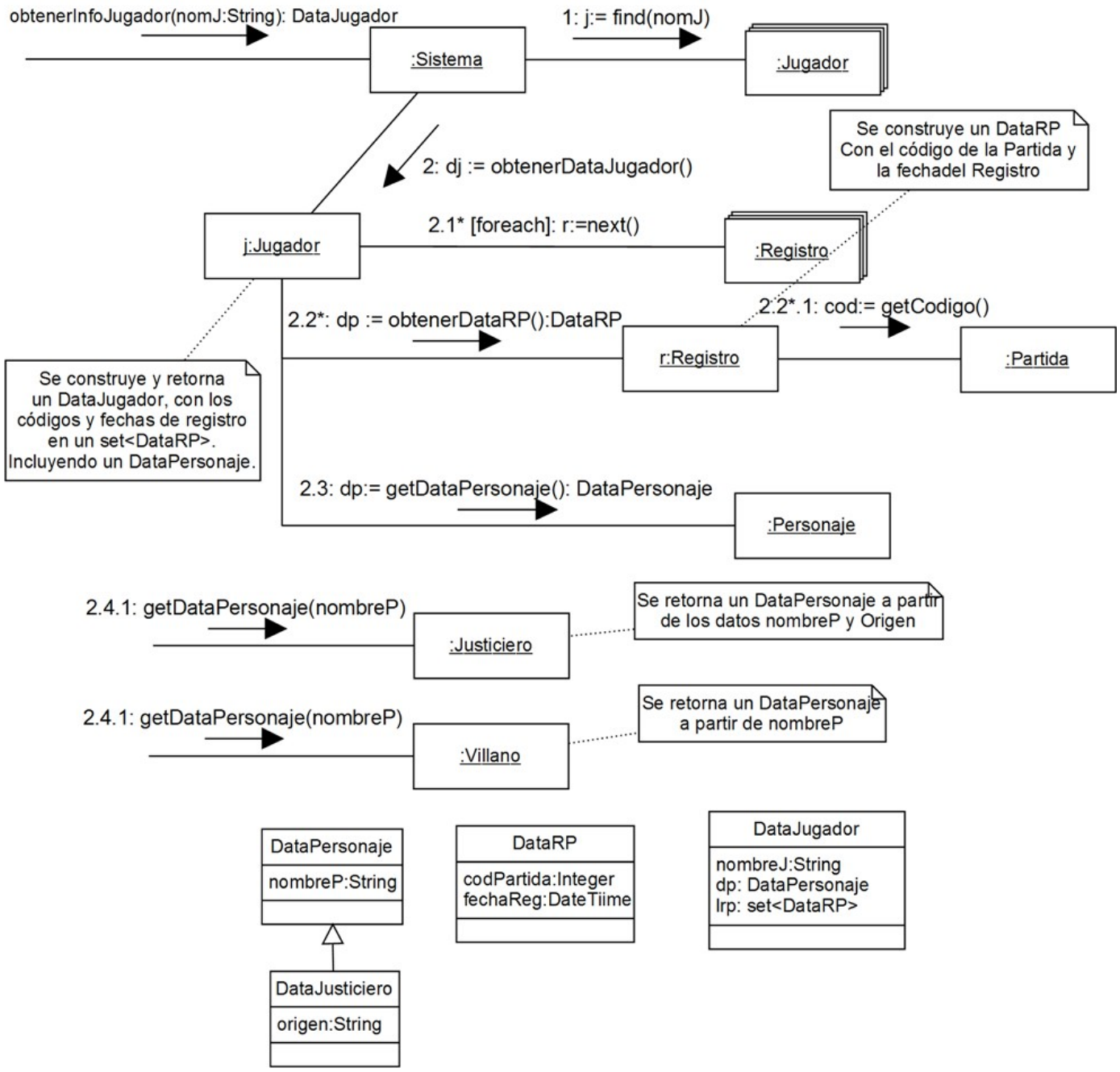
Parte A

Diagramas de comunicación

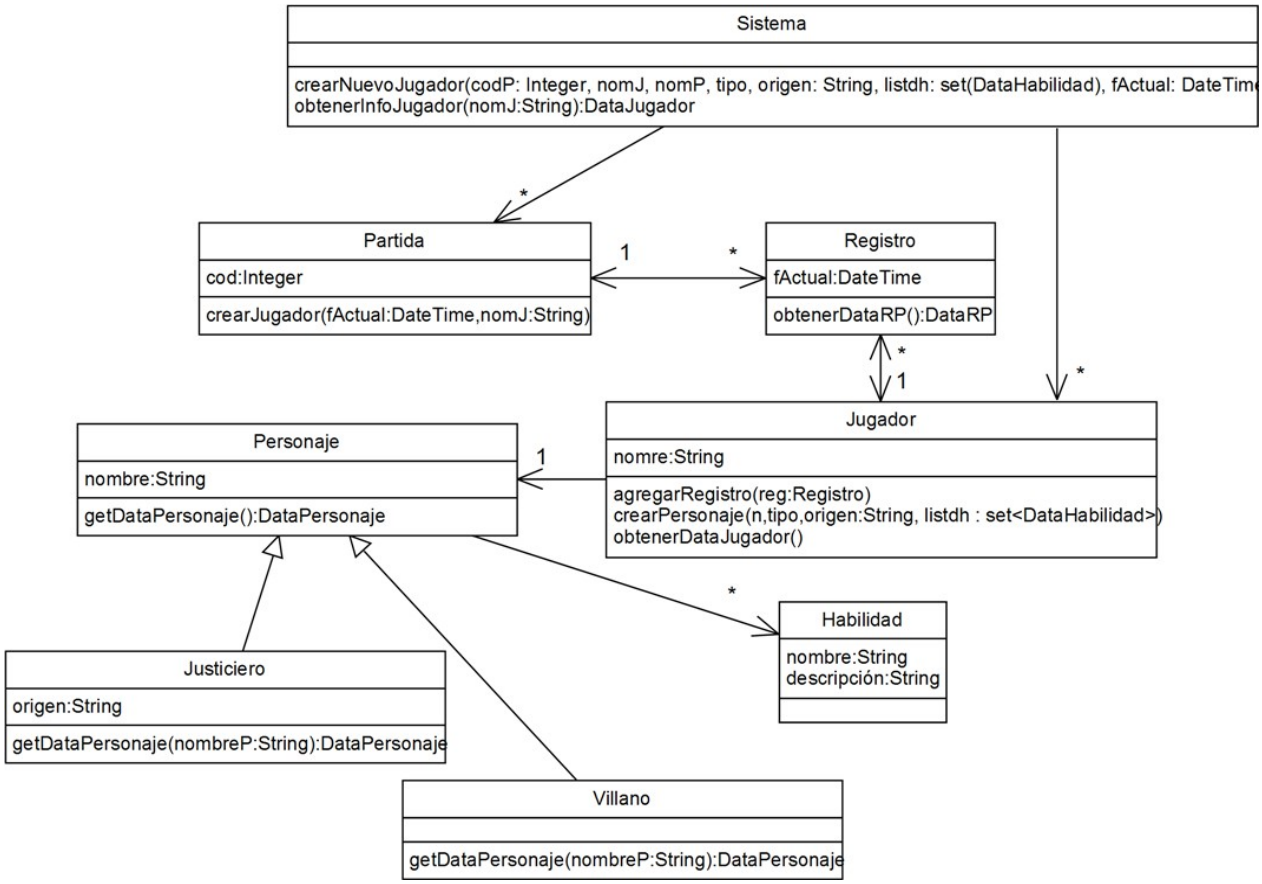
a. Crear nuevo Jugador



b. Obtener información de Jugador

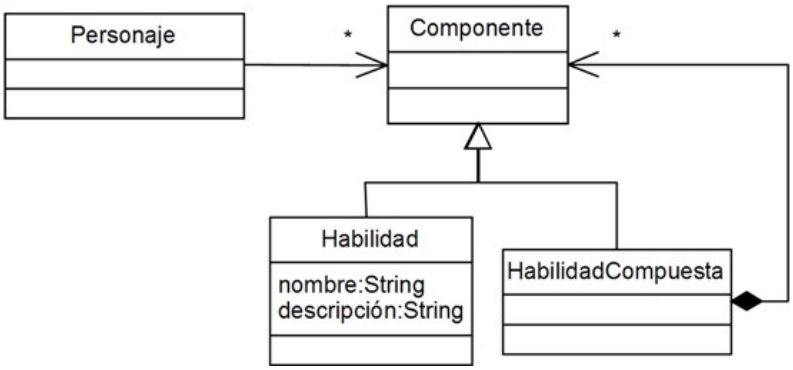


DCD



Parte B

Modificaciones al DCD



Patrón de Diseño

Patrón: Composite
Componente: Componente
Hoja: Habilidad
Compuesto: HabilidadCompuesta
Cliente: Personaje

Problema 3 (35 puntos)

Parte A

i.

```
class Prestamo : public ICollectible {
        private:
                int codigo;
                ItemPrestable *ip;
        public:
                Prestamo(int, ItemPrestable *);
                virtual ~Prestamo();
                int darCodigo();
                DataItemPrestable* darInfoItem();
                virtual DataPrestamo* darInfoPrestamo()=0;
};

class Sala : public Prestamo {
        private:
                int horaRetiro;
        public:
                Sala(int, ItemPrestable *, int);
                int darHoraRetiro();
                DataPrestamo *darInfoPrestamo();
};

class DataPrestamo : public ICollectible {
        private:
                int codigo;
                DataItemPrestable *dip;
        public:
                DataPrestamo(int, DataItemPrestable *);
                virtual ~DataPrestamo();
                int darCodigo();
                DataItemPrestable *darInfoItem();
};

class DataSala : public DataPrestamo {
        private:
                int horaRetiro;
        public:
                DataSala(int, DataItemPrestable *, int);
                int darHoraRetiro();
};
```

ii.

```
Prestamo::Prestamo(int c, ItemPrestable *i) {
        codigo = c;
```

```
	ip = i;
}

Prestamo::~Prestamo() {
}

Sala::Sala(int c, ItemPrestable *i, int h) : Prestamo(c,i) {
	horaRetiro = h;
}

DataPrestamo *Sala::darInfoPrestamo() {
	return new DataSala(darCodigo(), darInfoItem(), horaRetiro);
}

DataPrestamo::DataPrestamo(int c, DataItemPrestable *i) {
	codigo = c;
	dip = i;
}

DataPrestamo::~DataPrestamo() {
	delete dip;
}

DataSala::DataSala(int c, DataItemPrestable *i, int h) : DataPrestamo(c,i) {
	horaRetiro = h;
}
```

iii.

```
void Sistema::darPrestamos(ICollection *result) {
	IIterator *it = prestamos->getIterator();
	Prestamo *p;
	while (it->hasCurrent()) {
		p = dynamic_cast<Prestamo *>(it->getCurrent());
		result->add(p->darInfoPrestamo());
		it->next();
	}
	delete it;
	return result;
}
```

Parte B

iv.

```
class TablaIrregular {
	private:
		IDictionary *filas;
		int maxCols;
	public:
		TablaIrregular(int);
```

```
                ~TablaIrregular();
                int nuevaFila();
                void agregarColumna(int, String);
};

TablaIrregular::TablaIrregular(int mc) {
        filas = new ABB;
        maxCols = mc;
}

TablaIrregular::~TablaIrregular() {
        IIterator *it = filas->getIterator();
        ICollectible *i;
        while (it->hasCurrent()) {
                i = it->getCurrent();
                it->next();
                delete i;
        }
        delete it;
        delete filas;
}

int TablaIrregular::nuevaFila() {
        int cant = filas->size();
        cant = cant + 1;
        filas->add(new KeyInt(cant), new TablaRegular(maxCols));
        return cant;
}

void TablaIrregular::agregarColumna(int f, String v) {
        KeyInt *k = new KeyInt(f);
        TablaRegular *tr = filas->find(k);
        tr->agregarColumna(1,v);
        delete k;
}
```

v.
Se aplica el patrón Proxy.
Tabla es el Subject.
TablaRegular es el Real Subject.
TablaIrregular es el Proxy.