Paradigmas de la Programación – Segundo Parcial

11 de Junio de 2019

Apellido y Nombre: _____

1. **[10 pt.]** El algoritmo de linearización C3 se usa para obtener el orden en que se tienen que heredar los métodos en presencia de herencia múltiple, y se suele llamar Orden de Resolución de Métodos (Method Resolution Order (MRO)). Varios lenguajes de scripting, entre ellos Python y Perl (a partir de 6) lo implementan, tal como se muestra en el siguiente ejemplo:

```
class A {}
class B {}
class C {}
class D {}
class E {}
class K1 is A is B is C {}
class K2 is D is B is E {}
class K3 is D is A {}
class Z is K1 is K2 is K3 {}
say Z.^mro; # OUTPUT: ((Z) (K1) (K2) (K3) (D) (A) (B) (C) (E) (Any) (Mu))
```

   Explique por qué es necesario un algoritmo de linearización, qué problema resuelve y describa una estrategia más para resolver el mismo problema.

2. **[10 pt.]** En el siguiente fragmento de código, identifique características propias de los lenguajes de scripting, señálelas en el código y descríbalas.

```
float vlen(vector v) = #12;
entity nextent(entity e) = #47;

entity findnearestspawn(vector v)
{
    entity nearest;
    for (entity e = findchain(classname, "info_player_deathmatch"); e; e = e.chain
        if (!nearest) {
            nearest = e;
        } else if(vlen(e.origin - v) < vlen(nearest.origin - v)) {
            nearest = e;
        }
    }
    return nearest;
}
```

3. **[15 pt.]** En el siguiente fragmento de código vemos un actor que tiene semántica de acumulador sin usar asignación destructiva. El resultado de la acumulación se asigna por única vez al recolectar los resultados de todas las sumas que componen los cómputos parciales que realizaron otros actores.

   Identifique en el código construcciones lingüísticas con la siguiente semántica propia de actores:

   a) pasaje de mensajes,

   b) tipado de variables para excluir asignación destructiva,

   c) sincronización con otros procesos especificando variables con resultados parciales que serán satisfechos cuando se complete otro proceso.

```
val total = system.actorOf(Props[Total], "total")

val measurementsWebSocket =
  Flow[Message]
    .collect {
      case TextMessage.Strict(text) =>
        Future.successful(text)
      case TextMessage.Streamed(textStream) =>
        textStream.runFold("")(_ + _)
          .flatMap(Future.successful)
    }
    .mapAsync(1)(identity)
    .groupedWithin(1000, 1 second)
    .map(messages => (messages.last, Messages.parse(messages)))
    .map {
      case (lastMessage, measurements) =>
        total ! Increment(measurements.sum)
        lastMessage
    }
    .map(Messages.ack)

val route =
  path("measurements") {
    get {
      handleWebSocketMessages(measurementsWebSocket)
    }
  }

val bindingFuture = Http().bindAndHandle(route, "localhost", 8080)
```

4. En el siguiente texto se explica qué son los tipos nulificables, las excepciones de puntero nulo en Java y cómo Kotlin implementa estrategias para evitar este tipo de excepciones. Comente en sus propias palabras cuándo se da una excepción de puntero nulo en Java **[10 pt.]**.

   Explique cómo se relacionan las estrategias de Kotlin para evitar excepciones de puntero nulo con estrategias genéricas de programación defensiva **[10 pt.]**.

Java types are divided into primitive types (boolean, int, etc.) and reference types. Reference types in Java allow you to use the special value `null` *which is the Java way of saying "no object".*

A `NullPointerException` *is thrown at runtime whenever your program attempts to use a null as if it was a real reference. For example, if you write this:*

```
public class Test {
    public static void main(String[] args) {
        String foo = null;
        int length = foo.length();    // HERE
    }
}
```

the statement labelled "HERE" is going to attempt to run the `length()` *method on a null reference, and this will throw a* `NullPointerException`*.*

There're several strategies that can help us avoid this exception, making our codes more robust.

The most obvious way is to use `if (obj == null)` *to check every variable you need to use, either from function argument, return value, or instance field. When you receive a null object, you can throw a different, more informative exception like* `IllegalArgumentException`*. There are some library functions that can make this process easier, like* `Objects#requireNonNull`*.*

Kotlin was designed to eliminate the danger of null pointer references. It does this by making a null illegal for standard types, adding nullable types, and implementing shortcut notations to handle tests for null. For example, a regular variable of type String cannot hold null:

```
var a : String ="abc"
a = null // compilation error
```

If you need to allow nulls, for example to hold SQL query results, you can declare a nullable type by appending a question mark to the type, e.g. String?.

```
var b: String? ="abc"
b = null // ok
```

The protections go a little further. You can use a non-nullable type with impunity, but you have to test a nullable type for null values before using it.

To avoid the verbose grammar normally needed for null testing, Kotlin introduces a safe call, written `?.`*. For example,* `b?.length` *returns* `b.length` *if* `b` *is not null, and null otherwise.*

In other words, `b?.length` *is a shortcut for* `if (b != null) b.length else null`*.*

5. **[10 pt.]** En el siguiente fragmento de código, identifique construcciones lingüísticas con semántica de:

   a) sincronización de procesos

   b) manejo explícito de procesos

```
task = Task.async fn -> realizar_tarea_compleja() end
otra_tarea_larga()
Task.await task
```

6. **[10 pt.]** De estos dos fragmentos de código con funcionalidades comparables, uno tiene inversión de control y el otro no. Explique cuál es el que tiene inversión de control y justifique su respuesta.

```
interface CorrectorOrtografico {
    Arraylist<errores> CorreccionOrtografica(string Text);
}

Class CorrectorOrtograficoIngles : CorrectorOrtografico {
    public override Arraylist<errores> CorreccionOrtografica(string Text) {
        //la magia
    }
}

Class CorrectorOrtograficoFrances : CorrectorOrtografico {
    public override Arraylist<errores> CorreccionOrtografica(string Text) {
        //la magia
    }
}

Class TextEditor {
    CorrectorOrtografico objCorrectorOrtografico;
    string Text;
    public void TextEditor(CorrectorOrtografico objSC) {
        objCorrectorOrtografico = objSC;
    }

    public ArrayList <errores> CorreccionesOrtograficas() {
        return objCorrectorOrtografico.CorreccionOrtografica();
    }

}
```

```
Class TextEditor {
    //un monton de codigo para el editor

    CorrectorOrtograficoIngles objCorrectorOrtografico;
    String text;

    public void TextEditor() {
        objCorrectorOrtografico = new CorrectorOrtograficoIngles();
    }
    public ArrayList <errores> CorreccionOrtografica() {
        //devuelve errores de ortografia;
    }
}
```

7. **[15 pt.]** El siguiente fragmento de Elm explica cómo se tratan los errores en Elm. Describa con sus propias palabras esta política de tratamiento de errores y de su opinión sobre si esta política contribuye de alguna forma a la seguridad del lenguaje con respecto a vulnerabilidades por el sistema de tipos.

> *One of the guarantees of Elm is that you will not see runtime errors in practice. This is partly because Elm treats errors as data. Rather than crashing, we model the possibility of failure explicitly with custom types. For example, say you want to turn user input into an age. You might create a custom type like this:*

```
type MaybeAge
  = Age Int
  | InvalidInput

toAge : String -> MaybeAge
toAge userInput =
  ...

-- toAge "24" == Age 24
-- toAge "99" == Age 99
-- toAge "ZZ" == InvalidInput
```

> *Instead of crashing on bad input, we say explicitly that the result may be an Age 24 or an InvalidInput. No matter what input we get, we always produce one of these two variants. From there, we use pattern matching which will ensure that both possibilities are accounted for. No crashing!*

8. **[10 pt.]** A partir de la siguiente base de conocimiento, liste los subobjetivos que se tienen que satisfacer para comprobar que `digiriendo(gaviota,mosquito)`.

```
digiriendo(X,Y) :- comio(X,Y).
digiriendo(X,Y) :-
        comio(X,Z),
        digiriendo(Z,Y).

comio(mosquito,sangre(juan)).
comio(rana,mosquito).
comio(gaviota,rana).
```