Paradigmas de la Programación – Recuperatorio del Segundo Parcial

18 de Junio de 2019

Apellido y Nombre: ______________________________

1. **[10 pt.]** Ordene los siguientes fragmentos de código de más de scripting a menos de scripting, justificando su respuesta con al menos 3 características de los lenguajes de scripting.

```
main( ) {
  extrn a, b, c;
  putchar(a); putchar(b); putchar(c); putchar('!*n');
}
a 'hell';
b 'o, w';
c 'orld';
```

```
 puts "Hello, world!"
```

```
public class HelloWorld {
 %include { string.tom }
 public final static void main(String[] args) {
  String who = "world";
  %match(String who) {
   "World" -> { System.out.println("Hello, " + who + "!"); }
   _       -> { System.out.println("Don't panic"); }
  }
}
```

2. **[10 pt.]** En el siguiente texto se describe el uso de *streams* en programación reactiva. Explique en sus propias palabras qué es un *stream*, y cómo una componente declarativa puede tratar un stream, usando el concepto de "mensaje".

A stream is a sequence of ongoing events ordered in time. It can emit three different things: a value (of some type), an error, or a çompleted"signal. Consider that the çompleted"takes place, for instance, when the current window or view containing that button is closed.

We capture these emitted events only asynchronously, by defining a function that will execute when a value is emitted, another function when an error is emitted, and another function when 'completed' is emitted. Sometimes these last two can be omitted and you can just focus on defining the function for values. The "listening"to the stream is called subscribing. The functions we are defining are observers. The stream is the subject (or .ºbservable") being observed. This is precisely the Observer Design Pattern.

3. [**15 pt.**] El siguiente código no compila. Explique por qué, y qué habría que hacer para que compile.

```
class USBDevice
{
private:
    long m_id;

public:
    USBDevice(long id)
        : m_id(id)
    {
    }

    long getID() { return m_id; }
};

class NetworkDevice
{
private:
    long m_id;

public:
    NetworkDevice(long id)
        : m_id(id)
    {
    }

    long getID() { return m_id; }
};

class WirelessAdapter: public USBDevice, public NetworkDevice
{
public:
    WirelessAdapter(long usbId, long networkId)
        : USBDevice(usbId), NetworkDevice(networkId)
    {
    }
};

int main()
{
    WirelessAdapter c54G(5442, 181742);
    std::cout << c54G.getID(); // Which getID() do we call?

    return 0;
}
```

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 [**5 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. [**15 pt.**] El siguiente código es un ejemplo de programación defensiva. Identifique el mecanismo lingüístico con el que se implementa seguridad y sustitúyalo por otro con efectos equivalentes.

```
numbers = [1.5, 2.3, 0.7, -0.001, 4.4]
total = 0.0
for n in numbers:
    assert n >= 0.0, 'Assert 1'
    total += n
    print 'total is:', total
```

6. **[10 pt.]** En el siguiente fragmento de código, identifique construcciones lingüísticas con semántica de:

   a) sincronización de procesos

   b) manejo explícito de procesos

```
public class Counting {
    public static void main(String[] args) throws InterruptedException {
        class Counter {
            int counter = 0;
            public void increment() { counter++; }
            public int get() { return counter; }
        }

        final Counter counter = new Counter();

        class CountingThread extends Thread {
            public void  run() {
                for (int x = 0; x < 500000; x++) {
                    counter.increment();
                }
            }
        }

        CountingThread t1 = new CountingThread();
        CountingThread t2 = new CountingThread();
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.println(counter.get());
    }
}
```

7. **[10 pt.]** A partir de la siguiente base de conocimiento, liste los subobjetivos que se tienen que satisfacer para comprobar que `sabe_de(valen,julen)`.

```
sabe_de(X,Y) :- fue_presentado(X,Y).
sabe_de(X,Y) :-
        fue_presentado(X,Z),
        sabe_de(Z,Y).
fue_presentado(maxi,julen).
fue_presentado(sam,maxi).
fue_presentado(valen,sam).
```

8. **[15 pt.]** El segundo fragmento de código es equivalente al primero pero con inversión de control. Describa qué características del código convierten al segundo en un ejemplo de inversión de control, haciendo referencia a partes específicas del código.

```
class Superhero
{
    public $name;

    public function __construct(string $name)
    {
        $this->name = $name;
    }
}

class ComicBook
{
    public $mainCharacter;

    public function __construct(Superhero $superhero)
    {
        $this->mainCharacter = $superhero;
    }
}

$superhero = new Superhero('Caffeine Man');
$comic = new ComicBook($superhero);
```

```
interface CharacterInterface
{
    public function getName() : string;
}

abstract class Character implements CharacterInterface
{
    public $name;

    public function getName() : string
    {
        return $this->name;
    }
}

class Superhero extends Character
{
    public $name;

    public function __construct(string $name)
    {
        $this->name = $name;
    }
}

class VampiricDog extends Character
{
    public $name;

    public function __construct(string $name)
    {
        $this->name = $name;
    }
}

class ComicBook
{
    public $mainCharacter;

    public function __construct(CharacterInterface $character)
    {
        $this->mainCharacter = $character;
    }
}

$superhero = new Superhero('Caffeine Man');
$superheroComic = new ComicBook($superhero);

var_dump($superheroComic->mainCharacter->getName());

$vampireDog = new VampiricDog('Mr. Fangz');
$vampireComic = new ComicBook($vampireDog);

var_dump($vampireComic->mainCharacter->getName());
```