Examen final (formatif) - partie papier (1/2)

4 points - Code serveur

Voici des exemples d'annotations classiques côté serveur :

```
@PathVariable
@RequestBody
@RequestParam(name = "pipo")
@GetMapping("/chose/truc")
@ResponseBody
```

Écrivez le code d'un contrôleur serveur qui respecte les conditions suivantes :

- **2 points** Accepte des requêtes :
  - de méthode POST;
  - sur l'URL "/examen/{nom}/ajouterNote", où **nom** est une chaîne de caractères transmise dans l'URL;
  - reçoit dans le corps de la requête un objet de type **PostNoteRequest** (voir ci-dessous);
  - reçoit un entier obligatoire, nommé **bonus**, transmis comme paramètre de requête (ex: "/examen/{nom}/ajouterNote?bonus=3").
- **2 points** Renvoie une réponse HTTP :
  - additionne la note principale (**value**) et le bonus;
  - puis renvoie une réponse avec cette somme dans le corps.

```
public class PostNoteRequest {
    public Integer value;

    public PostNoteRequest(){
    }
}
```

Réponse :

```
@Controller
class Controleur {
    @PostMapping("/examen/{nom}/ajouterNote")
    public @ResponseBody ResponseEntity<Integer> ajouterNote(
   @PathVariable String nom,
   @RequestBody PostNoteRequest request
   @RequestParam(name="bonus") int bonus) {

        if (request.value == null) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
        }

        int somme = request.value + bonus
        return ResponseEntity.ok(somme)
    }
}
```

6 points - Trace client serveur

Soit le code suivant :

```
interface Service {
    @GET("longueur/{mot}")
    fun getLongueurMot(@Path("mot") mot: String): Call<Int>
}

object retrofit { // Ne pas inclure ce bloc de code dans la trace
    val service: Service = retrofit2.Retrofit.Builder()
        .addConverterFactory(retrofit2.converter.gson.GsonConverterFactory.create())
        .baseUrl("http://10.0.2.2:8080/")
        .build()
        .create(Service::class.java)
}

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val salutation = "hi"

        retrofit.service.getLongueurMot(salutation).enqueue(object : retrofit2.Callback<Int> {
            override fun onResponse(call: retrofit2.Call<Int>, response: retrofit2.Response<Int>) {
                if (response.isSuccessful) {
                    Toast.makeText(this@MainActivity, "Longueur: ${response.body()}", Toast.LENGTH_LONG).show()
                } else {
                    Toast.makeText(this@MainActivity, "Erreur: Mot trop court!", Toast.LENGTH_LONG).show()
                }
            }

            override fun onFailure(call: retrofit2.Call<Int>, t: Throwable) {
                Toast.makeText(this@MainActivity, "Erreur réseau", Toast.LENGTH_LONG).show()
            }
        })
    }
}
```

```
@Controller
class Controleur {
    @GetMapping("/longueur/{mot}")
    public @ResponseBody ResponseEntity<Any> getLongueur(@PathVariable String mot)  {
        if (mot.length >= 3) {
            return ResponseEntity.ok(mot.length);
        } else {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Mot trop court!");
        }
    }
}
```

Étant données les premières étapes suivantes, développez la trace d'exécution :

1. On part le serveur Spring Boot localement;
2. On démarre l'application Android;

Note : Vous pouvez utiliser des `[...]` pour éviter d'écrire les trop longues lignes de code, en autant que c'est clair de quelle ligne il s'agit, par exemple :

```
retrofit.service.getLongueurMot [...]
```

- **3 points** les lignes de code sont mentionnées dans l'ordre précis de leur exécution, il n'y a d'oubli ou de lignes en plus.
- **3 points** les colonnes effet et pile d'appels sont correctement remplies, par exemple les requêtes ou réponses mentionnent tous les éléments HTTP appropriés (méthode, corps etc.)