U.B.S. - I.U.T. de Vannes

Département Informatique

TP - Debugging

R2.03

Qualité de développement

BUT-Info-1A S2

R. Fleurquin, J.F. Kamp

Avril 2023

TP 1&2&3

Le rendu se fait en fin de troisième séance et contiendra dans un document `pdf` votre compte-rendu de l'exercice 3 uniquement :

- **Quelles erreurs vous avez pu détecter et en utilisant quelles techniques de debugging (variez les techniques de manière opportune).**
- **Votre proposition pour un code modifié respectant parfaitement les spécifications et commenté.**

Exercice 1

Ce premier exercice porte sur le code fournit ci-dessous. Vous devez le ressaisir sous Eclipse (notez que les deux classes sont dans le package `debugging`). Pendant toute la séance **on s'interdit d'apporter la moindre modification au code des deux classes utilisées**. Aucun affichage en particulier ne peut être ajouté ni aucune modification dans le `main()` des valeurs des variables et paramètres. Tout doit se faire via la perspective de debug et en particulier les vues *Variables*, *Breakpoints* et *Expressions*.

Question 1

En usant d'un seul *breakpoint* et de l'avancement de type *Step Over* déterminer à l'aide de la vue *Variables* les valeurs des variables `res1` et `res2`.

Question 2

Déterminez la valeur des variables `res1` et `res2` mais avec les valeurs respectivement 2, 3 et pour les variables `ba`, `exp`.

Question 3

Déterminez la valeur des variables `res1` et `res2` mais avec les valeurs respectivement 2, 2 et pour les variables `ba`, `exp` et la valeur 2 pour l'attribut `rang` de l'objet de type `Nombre`.

Question 4

Effacez tous vos *breakpoint* puis placez-en un en tout début du `main()`, puis un autre sur l'instruction de création de l'objet et enfin un troisième sur la dernière instruction du `main()`.

Lancez le debugging et avancez *breakpoint* par *breakpoint* en usant de l'avancement de type *Resume*.

Question 5

Supprimez depuis la fenêtre *Breakpoints* le dernier *breakpoint* (celui sur la dernière instruction). Relancez le debugging en usant de l'avancement de type *Resume*.

Desactivez tous les *breakpoints* (bouton *skip all*). Relancez le debugging. Que constatez-vous ?

Reactivez tous les Breakpoints (bouton *skip all*). Relancez le debugging en usant de l'avancement de type *Resume*.

Desactivez uniquement le second breakpoint. Relancez le debugging en usant de l'avancement de type *Resume*.

Question 6

Relancez un debugging et à l'aide de l'avancement de type *step into* allez dans le constructeur de l'objet de type `Nombre`. Observez le changement de contexte dans la fenêtre *Variables* et l'empilement du constructeur sur le `main()` dans la fenêtre de gauche de la *pile d'exécution*.

Avancez ensuite dans ce constructeur d'une seule instruction (mode *step over*) puis revenez directement au `main()` en un seul avancement de type *step return*.

Question 7

On veut maintenant utiliser le debugger pour comprendre la spécification de la méthode `calcul1()`.

Faites-en sorte que l'attribut `rang` de l'objet `Nombre` soit à 10 dans cette question 7 et 8 suivante à chacune de vos exécutions en mode debugging.

Placez un *breakpoint* dans la première instruction du corps de la boucle de cette méthode `calcul1()`. Lancez le debugging et relevez dans un tableau à chaque itération la valeur des variables `i`, `term` et `ret` en avançant en mode *step Resume*. Pouvez-vous déjà intuiter quelque chose sur ce qui est calculé ?

Pour en avoir le cœur net on se propose de comparer `ret` et `terme` avec une expression. On parle dans ce cas *d'invariant de boucle* en théorie de la programmation, c'est une propriété vraie à chaque itération d'une boucle. Si une telle expression est identifiée, elle permet pour certaine de déterminer formellement le résultat produit par une boucle et donc de « prouver mathématiquement » qu'une boucle s'il elle se termine produit bien le résultat attendu.

Dans la fenêtre Expressions, rajoutez les deux expressions :

- `ret==(1-Math.pow(this.exposant, i+1))/(1-this.exposant)`
- `terme==Math.pow(2,i)`

Constatez qu'à chaque fin d'itération (attention ne vous trompez pas sur l'état correspondant à cette fin !) ces deux expressions valent `true`. Elles sont donc bien des invariants de boucle. En déduire (avec vos vieux souvenirs de mathématiques sur les suite réelles de première) ce que calcul cette méthode.

Question 8

On veut maintenant comprendre ce que réalise la méthode `calcul2()`.

Désactivez le *breakpoint* placé dans `calcul1()`. Placez un *breakpoint conditionnel* dans le corps de la boucle de `calcul2()` qui ne se déclenche que pour `i=3` et `i=4`. Notez l'évolution de la variable `ret` entre ces deux itérations. Vous avez une piste ?

Placez une expression judicieusement choisie sur `ret` pour vérifier votre hypothèse. Cette expression dépendra de `base` et du paramètre `p`.

Concluez sur la spécification de `calcul2()`.

En déduire l'expression mathématique calculée par le `main()`. Vérifiez avec les questions 1, 2 et 3 que vous avez bien trouvé les bonnes spécifications pour les deux méthodes de calcul.

```java
package debugging;
public class Essai1 {

	public static void main(String[] args) {
		int ba=3;
		int exp =2;
		int res1;
		int res2;
		Nombre nb= new Nombre(ba,exp,3);

		res1=nb.calcul1();
		res2=nb.calcul2(res1);

	}
}
```

```java
package debugging;
public class Nombre {
	private int base;
	private int exposant;
	private int rang;

	public Nombre(int base, int exposant, int rg) {
		super();
		this.base = base;
		this.exposant = exposant;
		this.rang=rg;
	}

	public int calcul1() {
		int ret=1;
		int terme=1;
		for (int i=1;i<=this.rang;i++) {
			terme=terme*this.exposant;
			ret=ret+terme;
		}
		return ret;
	}

	public int calcul2(int p) {
		int ret=1;
		for (int i=0;i<p;i++) {
			ret=ret*this.base;
		}
		return ret;
	}

}
```

Exercice 2

On travaille avec les deux classes ci-dessous. Elles sont totalement indépendantes des classes précédentes. Supprimez tous vos *breakpoints* avant de commencer ce nouvel exercice.

Question 1

Mettez un *breakpoint* sur l'instruction `tab[i]=c1` et lancez l'exécution via le debugger. Quelle est la valeur de `i` pour cette exécution ? Regardez le contenu du tableau `tab` et retenez son identifiant Eclipse. Notez les identifiants et l'état des 3 objets de type `Carte` pointés par les variables `c1`, `c2` et `c3`.

À l'aide de la commande *Inspect* (clique droite de la souris sur le texte sélectionné des expressions Java à évaluer) déterminez la future valeur des deux expressions avec modulo : `(i+1)%3` et `(i+2)%3`.

Question 2

Depuis l'état d'exécution précédent, avancez instruction par instruction (mode *step over*) jusqu'à l'instruction de `swap` (que vous n'exécutez pas encore). Comparez alors les identifiants des deux tableaux `tab` et `lesCartes`. Que constatez-vous ? Pourquoi ? Notez les identifiants des objets dont l'adresse est stockée dans les 3 cases des tableaux. Prédisez avec un *Inspect* l'effet sur les tableaux de l'instruction de swap.

Exécutez alors le `swap`. Vérifiez votre prévision.

Question 3

Depuis l'état d'exécution précédent (vous devez être en attente d'exécution du `b.getUneCarte()`), à l'aide d'*Inspect*, prédisez l'état des 3 objets de type Carte après l'exécution du `swapColor()`.

Exécutez l'instruction et vérifiez vos prévisions.

Terminez l'exécution complète du `main()` et constatez le gain ou non de la partie.

Question 4

Supprimez tous vos *breakpoint*.

A l'aide uniquement d'un seul *breakpoint* bien placé et d'un seul *Inspect* et sans consultez l'état des variables, annoncez à l'avance si l'utilisateur va gagner ou perdre la partie que vous avez relancée.

Écrivez également une expression qui permettra de le savoir à la prochaine exécution.

Question 5

Supprimez tous vos *breakpoint*.

A l'aide uniquement d'un seul *breakpoint* bien placé et d'un seul changement de l'état d'une des variables (ou d'un *Inspect* avec un `setColor()`) garantissez le gain à coup sûr (en trichant) de l'utilisateur lors de l'exécution.

```java
package debugging;
public class Bonneteau {
	private Carte[] lesCartes;

	public static void main(String[] args) {
		Carte c1=new Carte("Rouge");
		Carte c2=new Carte("Verte");
		Carte c3=new Carte("Verte");
		Carte[] tab=new Carte[3];
		int i = (int)(Math.random()*3);
		tab[i]=c1;
		tab[(i+1)%3]=c2;
		tab[(i+2)%3]=c3;
		Bonneteau b=new Bonneteau(tab);
		b.swap(i,((i+i)%3));
		b.getUneCarte((5*i)%3).swapColor(c3);
		if(b.getUneCarte(0).getColor().equals("Rouge")) {
			System.out.println("Le 1 Gagne bravo !");
		}
		else {
			System.out.println("Le 1 perd désolé !");
		}
	}

	public Bonneteau(Carte[] lesCartes) {
		this.lesCartes = lesCartes;
	}

	public void swap(int i, int j) {
		Carte temp=this.lesCartes[i];
		this.lesCartes[i]=this.lesCartes[j];
		this.lesCartes[j]=temp;
	}

	public Carte getUneCarte(int k) {
		return this.lesCartes[k];
	}
}
```

```java
package debugging;
public class Carte {
	private String color;

	public Carte(String color) {
		super();
		this.color = color;
	}

	public String getColor() {
		return color;
	}

	public void setColor(String color) {
		this.color = color;
	}

	public void swapColor(Carte c) {
		String col=this.color;
		this.color=c.getColor();
		c.setColor(col);
	}
}
```

Exercice 3

On rajoute le code de la classe `Tableau` qui suit. La méthode `getMode()` doit retourner la (première s'il y a égalité) valeur qui est la plus fréquente dans un tableau non null et non vide et afficher avant cela le nombre maximum des occurrences trouvées.
Exemples :

- {1,3,3} Affiche « Le nombre maximum d'occurence est : 2 » retourne la valeur 3.
- {-1,2,3,2,3} Affiche « Le nombre maximum d'occurence est : 2 » pour la valeur 2.
- {} ou si null Affiche « Ce calcul est impossible »

Ce code ne fonctionne pas pour plusieurs raisons !

Uniquement avec le debugger, tentez de comprendre toutes les erreurs commises par le programmeur et proposez ensuite une **version totalement corrigée** de la méthode `getMode()`.

```
package debugging;

public class Tableau {
	private int[] tab;

	public static void main(String[]args) {
		int[] t= {4,1,1,3,4};
		Tableau tab=new Tableau(t);
		System.out.println(tab.getMode());
	}

	public Tableau(int[] tab) {
		super();
		this.tab = tab;
	}

	public int getMode() {
		int ret =-1;
		int nbMax=0;
		int nbOcc=0;
		if ((this.tab.length>0)&&(this.tab!=null)){
			ret=this.tab[0];
			for (int i=0;i<this.tab.length;i++){
				nbOcc=1;
				for (int j=i;j<=this.tab.length;j++){
					if(this.tab[i]==this.tab[j]){
						nbOcc++;
					}
				}
				if (nbOcc>=nbMax){
					nbMax=nbOcc;
					ret=i;
				}
			}
			System.out.println("Le nombre maximum d'occurence est :"+nbMax);
		}
		else{
			System.out.println("Ce calcul est impossible");
		}
		return ret;
	}
}
```