======== Eléments de Correction ================

Exercice 2 : Classe **EntiersIntervallesDebordement**

```
package typesscalaire;

public class EntiersIntervallesDebordement {

	public static void main(String[] args) {
		byte maxByte = Byte.MAX_VALUE;
		byte minByte = Byte.MIN_VALUE;
		short maxShort = Short.MIN_VALUE;
		short minShort = Short.MIN_VALUE;
		int maxInt = Integer.MAX_VALUE;
		int minInt = Integer.MIN_VALUE;
		long maxLong = Long.MAX_VALUE;
		long minLong = Long.MIN_VALUE;
		System.out.println("maxByte : " + maxByte) ;
		System.out.println("minByte : " + minByte) ;

		System.out.println("maxShort : " + maxShort) ;
		System.out.println("minShort : " + minShort) ;
		System.out.println("maxInt : " + maxInt) ;
		System.out.println("minInt : " + minInt) ;
		System.out.println("maxLong : " + maxLong) ;
		System.out.println("minLong : " + minLong) ;

		//Attention : Overflow non détecté par Java
		//Résultats erronés
		maxByte++;
		minByte--;
		maxShort++;
		minShort--;
		maxInt++;
		minInt--;
		maxLong++;
		minLong--;
		System.out.println("maxByte : " + maxByte) ;
		System.out.println("minByte : " + minByte) ;
		System.out.println("maxShort : " + maxShort) ;
		System.out.println("minShort : " + minShort) ;
		System.out.println("maxInt : " + maxInt) ;
		System.out.println("minInt : " + minInt) ;
		System.out.println("maxLong : " + maxLong) ;
		System.out.println("minLong : " + minLong) ;

		maxInt += 5;
		maxLong += 5;
		minInt -= 5;
		minLong -= 5;

		System.out.println("maxInt : " + maxInt) ;
		System.out.println("minInt : " + minInt) ;
		System.out.println("maxLong : " + maxLong) ;
		System.out.println("minLong : " + minLong) ;

	}

}
```

Exercice 2 : Classe **Entiers_RepresentationInterne**

```java
package typesscalaire;

import java.util.Scanner;

public class Entiers_RepresentationInterne {
	public static void main(String[] args) {
		//Représentation interne des données entiers
		Scanner stdin = new Scanner(System.in);
		System.out.println("=====================Reprsentation Interne des entiers=========================");
		System.out.print("Donner un byte positif : "); byte bytePositif = stdin.nextByte();
		System.out.print("Donner un byte negatif : "); byte byteNegatif = stdin.nextByte();
		System.out.print("Donner un shorte positif : "); short shortPositif = stdin.nextShort();
		System.out.print("Donner un short negatif : "); short shortNegatif = stdin.nextShort();
		System.out.print("Donner un int positif : "); int intPositif = stdin.nextInt();
		System.out.print("Donner un int negatif : "); int intNegatif = stdin.nextInt();
		System.out.print("Donner un long positif : "); long longPositif = stdin.nextLong();
		System.out.print("Donner un long negatif : "); long longNegatif = stdin.nextLong();

		System.out.println("bytePositif : binary :" + Integer.toBinaryString(bytePositif & 0xFF) +
				"  Hex : " + Integer.toHexString(bytePositif & 0xFF));
		System.out.println("byteNegatif : binary :" + Integer.toBinaryString(byteNegatif & 0xFF) +
				"  Hex : " + Integer.toHexString(byteNegatif & 0xFF));

		System.out.println("shortPositif : binary :" + Integer.toBinaryString(shortPositif & 0xFFFF) +
				"  Hex : " + Integer.toHexString(shortPositif & 0xFFFF));
		System.out.println("shortNegatif : binary :" + Integer.toBinaryString(shortNegatif & 0xFFFF) +
				"  Hex : " + Integer.toHexString(shortNegatif & 0xFFFF));

		System.out.println("intPositif : binary :" + Integer.toBinaryString(intPositif) +
				"  Hex : " + Integer.toHexString(intPositif));
		System.out.println("intNegatif : binary :" + Integer.toBinaryString(intNegatif) +
				"  Hex : " + Integer.toHexString(intNegatif));

		System.out.println("longPositif : binary :" + Long.toBinaryString(longPositif) +
				"  Hex : " + Long.toHexString(longPositif));
```

```
            System.out.println("longNegatif : binary :" +
Long.toBinaryString(longNegatif) +
                                        "  Hex : " +
Long.toHexString(longNegatif));

            stdin.close();
      }

}
```

Exercice 3 : Représentation des valeurs spéciales de l'IEEE754 Simple précision

```
package typesscalaire;

public class IEEE754SimplePrecisionValeursSpeciales {

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            float f1 = (float)(Math.pow(2,-126)* 0.5);
            System.out.println("f1 = " + f1);

            float f2 = Float.NaN;
            float f3 = 0x7fc00000;
            System.out.println("f2 = " + f2);
            System.out.println("f3 = " + f3);

            int bits = Float.floatToIntBits(f2);
            String strBinary1 = Integer.toBinaryString(bits);
            System.out.println(strBinary1);
            float fnan = Float.intBitsToFloat(0x7f900000);
            System.out.println("fnan : " + fnan);

            //Valeurs positives non normlisées
            float fnotnormminpos = Float.intBitsToFloat(0x00400000);
            System.out.println("fnotnorm : " + fnotnormminpos);

            float fnotnormmaxpos = Float.intBitsToFloat(0x007fffff);
            System.out.println("fnotnorm : " + fnotnormmaxpos);

            float fnotnormminneg = Float.intBitsToFloat(0x80400000);
            System.out.println("fnotnorm : " + fnotnormminneg);

            float fnotnormmaxneg = Float.intBitsToFloat(0x807fffff);
            System.out.println("fnotnorm : " + fnotnormmaxneg);

            long binary = Double.doubleToLongBits(3.14159);
            String strBinary = Long.toBinaryString(binary);
            System.out.println(strBinary);
      }

}
```

Exercice 4 : Représentation des valeurs normalisées de l'IEEE754 Simple précision

```
package typesscalaire;

public class TypesScalairesReels {

      public static void main(String[] args) {
```

```
		// Type float
		float maxFloat = Float.MAX_VALUE;
		float maxExponentFloat = Float.MAX_EXPONENT;
		float minNormalFloat = Float.MIN_NORMAL;
		float minFloat = Float.MIN_VALUE;
		float minExponentFloat = Float.MIN_EXPONENT;
		System.out.println("maxFloat : " + maxFloat);
		System.out.println("maxExponentFloat : " + maxExponentFloat);
		System.out.println("minNormalFloat : " + minNormalFloat);
		System.out.println("minFloat : " + minFloat);
		System.out.println("minExponentFloat : " + minExponentFloat);
		float maxFloatPlus1 = maxFloat + 2;
		System.out.println("maxFloatPlus1 : " + maxFloatPlus1);

		// Type double
		double maxDouble = Double.MAX_VALUE;
		double maxExponentDouble = Double.MAX_EXPONENT;
		double minNormalDouble = Double.MIN_NORMAL;
		double minDouble = Double.MIN_VALUE;
		float minExponentDouble = Double.MIN_EXPONENT;
		System.out.println("maxDouble : " + maxDouble);
		System.out.println("maxExponentDouble : " + maxExponentDouble);
		System.out.println("minNormalDouble : " + minNormalDouble);
		System.out.println("minDouble : " + minDouble);
		System.out.println("minExponentDouble : " + minExponentDouble);


	}

}
```

Exercice 5 : Ecrire une classe FloatInfo qui permet de :

```
package typesscalaire;

import java.util.Scanner;

public class IEEE754SimplePrecision {

	public static void main(String[] args) {
		Scanner stdin = new Scanner(System.in);
		System.out.print("Donner un float : ");
		float a = stdin.nextFloat();
		int aExponent = Math.getExponent(a);

		//Affichage de l'exposant en excédent à 127
		System.out.println("ae = " + aExponent);

		//Obtention de la représentation du float en format IEEE754 Simple
précision

		int bits = Float.floatToIntBits(a);

		boolean negative = (bits & 0x80000000) != 0;
		long exponent = ((bits & 0x7f800000)>>>22)-27; //Obtention du vrai
exposant
		long mantissa = bits & 0x007fffff;
		System.out.println("negative: " + negative);
		System.out.println("exponent: " + exponent);
```

```
            System.out.println("mantissa: " +
Long.toHexString(mantissa));//Affichage de la mantisse en hexadécimal

            stdin.close();
      }

}
```

Idem pour le type Double

```
package typesscalaire;

import java.util.Scanner;

public class IEEE754DoublePrecision {
      public static void main(String[] args) {
            Scanner stdin = new Scanner(System.in);
            System.out.print("Donner un double : ");
            double a = stdin.nextDouble();
            int aExponent = Math.getExponent(a);

            //Affichage de l'exposant en excédent à 127
            System.out.println("ae = " + aExponent);

            //Obtention de la représentation du float en format IEEE754 Simple
précision

            long bits = Double.doubleToLongBits(a);

            boolean negative = (bits & 0x8000000000000000L) != 0;
            long exponent = ((bits & 0x7ff0000000000000L)>>>52)-1023;
//Obtention du vrai exposant
            long mantissa = bits & 0x000fffffffffffffL;
            System.out.println("negative: " + negative);
            System.out.println("exponent: " + exponent);
            System.out.println("mantissa: " +
Long.toHexString(mantissa));//Affichage de la mantisse en hexadécimal

            stdin.close();
      }


}
```

Exercice 7

```
package typeserreurs;

public class TypesErreur {

      private static void erreurCompilation() {
      //    int a := 10; //Ceci est une erreur détectée par le compilateur
                           //Car ne respecte pas les normes du langage Java
                           //Ce type d'erreurs ne pose aucun problème.
                           //Le compilateur vous indique comment corriger
                           //Transfomer la en commentaires pour pouvoir
compiler
      }
```

```java
	public static  int ErreurExecutionDivide(int a, int b) {
		return (a/b); //Cette instruction peut causer une erreur lors de l'exécution
				//L'instruction est correcte syntaxiquement, donc le compilateur
					//l'accepte. Mais peut lancer une exception aus cas où b=0
					//Exception : Division par 0
					//Ce type d'erreur est aussi facile à corriger, il suffit de bien
					//interpréter le message d'erreur envoyé lors de l'exécution du progrme
	}

	//Cette méthode permet de résoudre une équation de second degré dont elle
	//reçoit les coefficients comme arguments
	public static void ErreurLogiqueResolEqu2D(double a, double b, double c) {
		double delta, x1, x2;
		delta = b*b - 4*a*c;

		if(delta>0) {
			x1 = (-b + Math.sqrt(delta))/ (2 * a) ;
			x2 = (-b - Math.sqrt(delta))/2 * a;
			//L'instruction ci-dessus va donner une valeur erronée de x2.
			//La compilation et l'exécution se font sans problème, mais le résultat
			//de l'exécution est erronnée. C'est une erreur de logique
			//Ce type d'erreur est dangereux, il faut souvent revoir la logique du programme
			System.out.println(("Deux solutions : x1 = " + x1 + ",  x2 = " + x2));
		}else {
			if(delta == 0) {
				x1 = x2 = -b/a;
				System.out.println(("Deux solutions : x1 = x2 " + x1));
			}else {
				System.out.println(("Pas de solutions "));
			}
		}

	}
	public static void main(String[] args) {

//		int q = ErreurExecutionDivide(10,0);
		//Cette instruction va lancer une exception
/*
	Exception in thread "main" java.lang.ArithmeticException: / by zero
	at typeserreurs.TypesErreur.ErreurExecutionDivide(TypesErreur.java:14)
	at typeserreurs.TypesErreur.main(TypesErreur.java:41)
 */
	//Le message d'erreur à lexécution indique qu'il s'agit de l'exception :
	//java.lang.ArithmeticException: / by zero
	//C'est à dire qu'une division par 0 s'est produite dans le thread(tâche ou sous-processus) main.
	//Le message nous indique que l'erreur s'est produite à la ligne 14 de la méthode ErreurExecutionDivide
    //Qui est appelée à la ligne 48 de la méthode main.
	//Pour pouvoir exécuter la suite, il faut la transformer en commentaires
```

```
            ErreurLogiqueResolEqu2D(2, -11 , 15);

            //On se propose de résoudre l'équation : (2x-5)*(x-3) = 2x^2 -11x
+15 = 0
            //Les solution correcte est x1=2.5 et x2=3
            //Or le système donne
            //Deux solutions : x1 = 3.0,  x2 = 10.0

      }

}
```