Nom : .....

Prénom : .....

Promo : .....

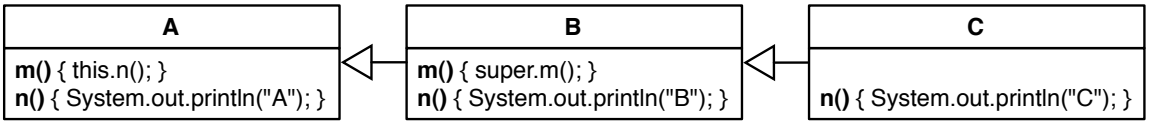
Toutes les réponses sont courtes si vous vous contentez de répondre à la question posée, sans commentaires sauf s'ils sont explicitement demandés. Répondre sur l'énoncé à priori, sur copie en cas de problème.

Seuls les polycopiés distribués en cours sont autorisés.

1. Quel est le rôle d'une classe par rapport à ses instances ?

   Solution: Une classe :
   1. spécifie le format et le comportement de ses instances (nom et type des variables d'instance, méthodes),
   2. est capable de créer des nouvelles instances.

2. Dans le contexte de ces trois classes :



   Ce code Java...

```
B b = new C();
b.m();
```

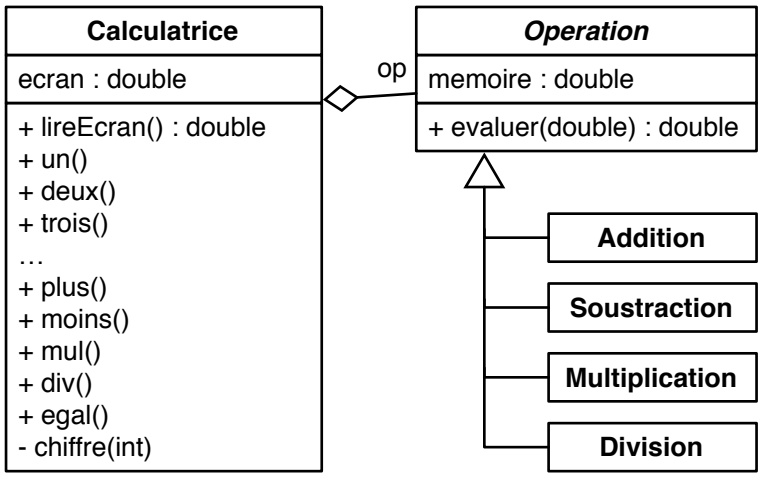
   A. affiche `A` B. affiche `B` *C. affiche `C`* D. ne compile pas E. boucle sans fin

3. On considère l'étude de cas des fonctions mathématiques, où `Function` est l'interface générale des fonctions, `NullaryFunction`, `UnaryFunction` et `BinaryFunction` sont trois classes abstraites définissant le nombre d'arguments de la fonction et leur représentation en Java, et `Val`, `Sin`, `Plus` sont des sous-classes concrètes (de `NullaryFunction`, `UnaryFunction`, `BinaryFunction`, respectivement) qui représentent des fonctions connues.

   Cochez si les instructions suivantes sont correctes ou pas, selon le compilateur Java (en supposant que les arguments `x` ou `y` des constructeurs sont à chaque fois du bon type) :

Correct	Incorrect	Expression
☐	☑	`Function f = new Function();` *interface, pas instanciable*
☑	☐	`Plus f = new Plus(x,y);`
☑	☐	`UnaryFunction f;` *juste une déclaration*
☐	☑	`UnaryFunction f = new UnaryFunction(x);` *une classe abstraite n'est pas instanciable*
☑	☐	`UnaryFunction f = new Sin(x);` *Sin est une sous-classe de UnaryFunction*
☐	☑	`UnaryFunction f = new Val(x);` *Val n'est pas une sous-classe de UnaryFunction*
☑	☐	`Object f = new Plus(x,y);` *Object super-classe de Plus...*
☐	☑	`Plus f = new Object();` *...pas l'inverse*

4. On veut implémenter une calculatrice "quatre opérations" suivante :



En plus de la méthode `lireEcran()`, la classe calculatrice a une méthode par touche. Il y a trois sortes de méthodes-touche :

– les chiffres modifient le nombre affiché (pour simplifier, on ne s'intéresse pas à la frappe des nombres à virgule) ;
– les opérateurs créent et mémorisent dans `op` un objet `Operation` correspondant à l'opération arithmétique à effectuer et son premier argument, et réinitialisent l'écran pour le deuxième argument ;
– la méthode `egal()` évalue l'opération en cours et met à jour l'écran.

Le programme de test suivant illustre l'usage envisagé de la calculatrice :

```
public static void main(String[] args) {
    Calculatrice c = new Calculatrice();
    c.trois();
    c.plus();
    c.un();
    c.huit();
    c.egal(); // 3 + 18 = ...
    System.out.println(c.lireEcran()); // ... 21 !
}
```

(a) Implémentez la méthode `chiffre(int n)` qui met à jour le nombre affiché à l'écran avec le chiffre additionnel `n`. Le but est de factoriser le comportement pour les touches chiffres :

```
public void zero() { this.chiffre(0); }
public void un() { this.chiffre(1); }
// etc
```

Solution:

```
protected void chiffre(int n) {
    ecran = ecran * 10 + n;
}
```

(b) Implémentez la méthode `egal()` qui combine le premier nombre et l'opérateur représentés par `op`. Considérez ce qu'il doit se passer si `op == null` (si on tape `=` sans opérateur).

Solution:

```
public void egal() {
    if (op != null) {
        ecran = op.evaluer(ecran);
    }
}
```

(c) Les quatre méthodes-opérateurs sont quasi-identiques. Implémentez-en une et indiquez que ce qui diffère pour les trois autres. Pensez à l'enchaînement d'opérations :

`3` → affiche 3,
`+` → mémorise `new Addition(3)`, affiche 0,
`18` → affiche 18,
`*` → mémorise `new Multiplication(21)`, affiche 0,
`2` → affiche 2,
`=` → affiche 42.

Solution: Un opérateur termine le nombre en cours d'entrée, et mémorise l'opération en attente du second opérande. On appelle `calcule()` pour évaluer d'abord une éventuelle opération déjà en cours (p.ex. quand on appuie sur `*`, c'est le résultat de l'addition qui devient l'opérande gauche de la multiplication). En fait, les opérateurs se comportent comme si on appuyait systématiquement sur `=` avant.

```
public void plus() { // idem moins() mul() div()
    this.egal();
    op = new Addition(ecran); // ou new Soustraction(ecran)... etc
    ecran = 0; // on a mémorisé l'op
}
```

(d) Implémentez la classe abstraite `Operation` et ses sous-classes. Les quatre sous-classes étant très similaires, n'en implémentez qu'une complètement, et indiquez juste ce qui diffère dans les trois autres.

Solution:

```
public abstract class Operation {

    protected double memoire;

    public Operation(double arg) {
        memoire = arg;
    }
    public abstract double evaluer(double arg);
}
```

Solution:

```
public class Addition extends Operation {

    public Addition(double arg) {
        super(arg);
    }

    public double evaluer(double arg) {
        return memoire + arg;
    }
}
```

(e) Donnez le principe des modifications à effectuer pour gérer :
- la touche "point" et les nombres à virgule.
- les touches unaires agissant sur le nombre affiché, comme le changement de signe (touche `+/-`), x^2, $\sqrt{x}$, ou $1/x$;

Solution: Une fois qu'on a pressé la touche "point", les chiffres ajoutent une valeur de plus en plus petite (les décimales, de gauche à droite). Il faut donc mémoriser l'appui sur "point" et la valeur de la décimale à ajouter, rendre le comportement de `chiffre(n)` conditionnel, et (ré)initialiser les variables (appui sur un opérateur ou la touche égal).

Une explication suffisait, mais voici à quoi le code pourrait ressembler :

```
public class CalculatriceDecimale extends Calculatrice {

    protected boolean entreeDecimales;
    protected double valeurDecimale;

    public CalculatriceDecimale() {
        super();
        razEntreeDecimales();
    }

    private void razEntreeDecimales() {
        entreeDecimales = false;
        valeurDecimale = 1.0;
    }

    public void point() { entreeDecimales = true; }

    public void chiffre(int n) {
        if (entreeDecimales) { // nouveau cas
            valeurDecimale = valeurDecimale / 10.0;
            ecran = ecran + (n * valeurDecimale);
        } else { // sinon c'est un chiffre normal
            super.chiffre(n);
        }
    }

    public void egal() {
        razEntreeDecimales();
        super.egal();
    }
}
```

Pour gérer les touches unaires, on peut modifier directement le nombre à l'écran :

```
public void oppose() { ecran = -ecran; }
public void carre() { ecran = ecran * ecran; }
public void racine() { ecran = Math.sqrt(ecran); }
public void inverse() { ecran = 1.0 / ecran; }
```