Exercice 1 : itérateur (5 points)

a) Donner le code complété de la classe IterateurAFaire (4 points).

```
private class IterateurAFaire implements Iterator<ChoseAFaire> {
 private Iterator<ChoseAFaire> iter;
 private ChoseAFaire courante;
 public IterateurAFaire(Iterator<ChoseAFaire> it) {
  iter = it;
 }
 public boolean hasNext() {
  while (iter.hasNext()) {
   courante = iter.next();
   if (!courante.fini()) return true;
  }
  return false;
 }
 public ChoseAFaire next() {
   return courante;
 }
}
```

b) Donner la nouvelle version de lméthode iterator (1 point).

```
public Iterator<ChoseAFaire> iterator() {
  if (tout)
    return contenu.iterator();
  else
    return new IterateurAFaire(contenu.iterator());
}
```

Exercice 2 : Visiteur (10 points)

a) Donner le code de la classe ChargementV1 (4 points).

```
public class ChargementV1 implements Operation {
      private DataInputStream s;
      public ChargementV1(DataInputStream s) {
            this.s = s;
      }
      public void opereSur(ChoseAFaire c) throws IOException {
            c.nom = s.readUTF();
            c.fini = false;
      }
      public void opereSur(PenseBete p) throws Exception {
            int n = s.readInt();
            for (int i = 0; i < n; i++) {
                  ChoseAFaire c = new ChoseAFaire();
                  c.realise(this);
                  p.ajoute(c);
            }
      }
}
```

b) Donner le code de la classe ChargementV2 (4 points).

```
public class ChargementV2 implements Operation {
	private DataInputStream s;
	public ChargementV2(DataInputStream s) {
		this.s = s;
	}
	public void opereSur(ChoseAFaire c) throws IOException {
		c.nom = s.readUTF();
		c.fini = s.readBoolean();
	}
	public void opereSur(PenseBete p) throws Exception {
		int n = s.readInt();
		for (int i = 0; i < n; i++) {
			ChoseAFaire c = new ChoseAFaire();
			c.realise(this);
			p.ajoute(c);
		}
	}
}
```

c) Donner le code de la version modifiée de la méthode charge de la classe PenseBete (2 points).

```
	public OuverturePenseBete charge(String nomFichier) throws Exception {
	Operation chargement;
	try (DataInputStream s = new DataInputStream(
					new FileInputStream(
						new File(nomFichier)))) {
		vide();
		byte version = s.readByte();
		switch (version) {
			case 1:
				chargement = new ChargementV1(s);
				break;
			case 2:
				chargement = new ChargementV2(s);
				break;
			default:
				chargement = null;
		}
		if (chargement != null) {
			this.realise(chargement);
		} else {
			throw new ExceptionVersionInconnue(version);
		}
	} catch (ExceptionVersionInconnue eVersion) {
		return OuverturePenseBete.VERSION_INCONNUE;
	} catch (IOException eAutre) {
		return OuverturePenseBete.PROBLEME_PENDANT_LECTURE;
	}
	return OuverturePenseBete.REUSSIE;
}
```

Exercice 3 : Composite (5 points)

a) Donner le code de la méthode fini() de la classe ChoseAFaireComplexe (1,5 points).

```
public boolean fini() {
	for (ChoseAFaire c : contenu) {
		if (!c.fini()) return false;
	}
	return true;
}
```

b) Donner le code de la méthode termine() de la classe ChoseAFaireComplexe (1,5 points).

```
public void termine() {
	for (ChoseAFaire c : contenu) {
		c.termine();
	}
}
```

c) Donner le code de la méthode toString () de la classe ChoseAFaireComplexe (2 points).

```
protected String toString(String marge) {
	String texte = marge;
	texte += (fini()) ? "X " : "  ";
	texte += nom + "\n";
	for (ChoseAFaire c : contenu) {
		texte += c.toString(marge + " ") + "\n";
	}
	return texte;
}
public String toString() {
	return toString("");
}
```

Hypothèses :
1) La classe ChoseAFaire comprend la déclaration suivante :

```
protected abstract String toString(String marge);
```

2) La classe ChoseAFaireSimple comprend les définitions suivantes :

```
protected String toString(String marge) {
	String texte = marge;
	texte += (fini) ? "X " : "  ";
	texte += nom;
	return texte;
}
public String toString() {
	return toString("");
}
```