7. In our rich class hierarchy the class **Apple** has following subclasses:

```
class GoldenDelicious extends Apple {}
class Jonagold extends Apple {}
```

Our fruit processing application contains a utility class which can decide, whether an apple is ripe:

```
class FruitHelper {
        public static boolean isRipe(Apple apple) {
                ...
        }
}
```

In the class **FruitHelper** we want to implement a method which can look into any basket which can contain apples only and decide, whether the apple in the basket is ripe of not. Here the body of the method:

```
{
        Apple apple = basket.getElement(); // 1
        return isRipe(apple); // 2
}
```

What should the signature of the method look like:

a) ☐
```
public static boolean
isRipeInBasket(Basket basket)
```

b) X
```
public static boolean
isRipeInBasket(Basket<Apple> basket
```

c) ☐
```
public static boolean
isRipeInBasket(Basket<?> basket)
```

d) ☐
```
public static boolean
isRipeInBasket(Basket<? extends Apple> basket)
```

e) ☐
```
public static <A extends Apple> boolean
isRipeInBasket(Basket<A> basket)
```

f) ☐
```
public static <A> boolean
isRipeInBasket(Basket<A extends Apple> basket)
```

g) ☐
```
public static boolean
isRipeInBasket(Basket<T super Apple> Basket)
```

8. Now we want to implement a method which inserts only ripe apples into the basket. Here the method's body:

```
{
        if (isRipe(apple)) { // 1
                basket.setElement(apple); // 2
        }
}
```

Which of these signatures should we use?

a) X
```
public static void
insertRipe(Apple apple, Basket<Apple> basket)
```

b) ○
```
public static void
insertRipe(Apple apple, Basket<? extends Apple> basket)
```

c) ○
```
public static void
insertRipe(Apple apple, Basket<? super Apple> basket)
```

d) ○
```
public static <A extends Apple> void
insertRipe(A apple, Basket<? super A> basket)
```

e) ○
```
public static <A super Apple> void
insertRipe(A apple, Basket<? extends A> basket)
```

9. We could acquire some expertise in the *orangeology* and now we can decide whether an orange is ripe or not - and this in pure Java. Now we want to extend the class **FruitHelper**.

Here is our updated source code :

```
class FruitHelper {
	public static boolean isRipe(Apple apple) {
		... // censored to protect our know-how
	}

	public static boolean isRipe(Orange orange) {
		... // censored to protect our know-how
	}

	public static boolean isRipeInBasket(Basket<? extends Apple>
basket) {
		Apple apple = basket.getElement();
		return isRipe(apple);
	}

	public static boolean isRipeInBasket(Basket<? extends Orange>
basket) {
		Orange orange = basket.getElement();
		return isRipe(orange);
	}
}
```

Is this source code OK?

a) ○ Yes. The source code is OK.

b) X No. The source code cannot be compiled.

10. What about the following source code. Can it be compiled?

```
class FruitHelper {
	public static boolean isRipe(Apple apple) {
		... // censored to protect our know-how
	}

	public static boolean isRipe(Orange orange) {
		... // censored to protect our know-how
	}

	public static <A extends Apple>
	void insertRipe(A a, Basket<? super A> b)
	{
		if (isRipe(a)) {
			b.setElement(a);
		}
	}

	public static <G extends Orange>
	void insertRipe(G g, Basket<? super G> b)
	{
		if (isRipe(g)) {
			b.setElement(g);
		}
	}
}
```

a) ○ Yes. The source code is OK.

b) X No. The source code cannot be compiled.

11. The accounting department needs to know how many baskets we produce. So we've changed the class **Basket**:

```
public class Basket<E> {

  ...

        private static int theCount = 0;
        public static int count() {
                return theCount;
        }

        Basket() {
                ++theCount;
        }

        ...
}
```

What output would be produced by the following source code?

```
public static void main(String[] args) {
        Basket<Apple> bA = new Basket<Apple>();
        Basket<Orange> bG = new Basket<Orange>();
        System.out.println(bA.count());
}
```

a) ○ 1

b) X 2

c) ○ Compile error

12. What abut the following source code?

```
Basket<Orange> bG = new Basket<Orange>(); // 1
Basket b = bG; // 2
Basket<Apple> bA = (Basket<Apple>)b; // 3
bA.setElement(new Apple()); // 4
Orange g = bG.getElement(); // 5
```

a) ○ No compile error, no exception during the runtime

b) ○ Compile error in the line 3

c) X **ClassCastException** the line 3

d) ○ Compile warning in the line 3, **ClassCastException** in the line 4

e) ○ Compile warning in the line 3, **ClassCastException** in the line 5

13. And what about this one?

```
Basket<Orange> bG = new Basket<Orange>(); // 1
Basket<? extends Fruit> b = bG; // 2
if (b instanceof Basket<Apple>) { // 3
        Basket<Apple> bA = (Basket<Apple>) b; // 4
        bA.setElement(new Apple()); // 5
} // 6
Orange g = bG.getElement(); // 7
```

a) ○ No compiler error, no exception

b) ○ Compiler error in the line 3

c) X Compiler warning in the lines 3 and 4. An exception will be thrown in the line 7.