Access Modifiers – protected access

Learning Outcomes:

After doing this exercise, the learner…

1. **has been confronted with common protected-access problem across packages**
2. **has had to recall detail about inheritance**
3. **has had to think deeply about import and package statements and inheritance in Java**

Task:

Copy and compile the codes below yourself and experiment briefly to discover what each scenario teaches (oh!, and ask questions).

Scenario 1: Protected Access – Fiddle and PlainFiddle

Study the code below – a programmer is trying to create the class `ClassicFiddle` such that it inherits the `play()` method from the class `PlainFiddle`. The `PlainFiddle` class and the `ClassicFiddle` class are however in different packages and the programmer is having difficulty getting the code to compile.
She has searched the internet and added a post to a forum where she got a tip/hint that she needs to use the `protected` keyword to get the code to compile. Use the tip/hint and fix the code below.

```
package plain;

class PlainFiddle{
     void play(){
          System.out.println(this+" play()");
     }
}
```

```
package classic;               //different package

class ClassicFiddle {          //anything missing here?
     void playSweetly(){
          play();              //attempt access through inheritance
          System.out.println(this+" playSweetly()");
     }

     public static void main(String[] args){
          ClassicFiddle c = new ClassicFiddle();
          c.playSweetly();
     }

}
```

Changes Required:
In class PlainFiddle.java:
1.
2.
In class ClassicFiddle.java
1.
2.

Compile statements:
1.
2.
Execute statement:
1.

Scenario 2: More Complex Protected Access

Making a member of a class `protected` allows access to that member

1. by a class in the same package **OR**,
2. through inheritance (i.e. a class can refer to it as if it were its own member) – it does not allow access through a reference. Compile each of the following :

```
package plain;

public class PlainFiddle{
                         //play(): protected(#)
     protected void play(){
          System.out.println("play(): "+this);
     }
}
```

```
package plain;              //same package

class SamePackageClass{

     public static void main(String[] args){
          PlainFiddle p = new PlainFiddle();
          p.play();         //works! package access
     }
}
```

```
package classic;            //different package
import plain.PlainFiddle;

class ClassicFiddle extends PlainFiddle{
     void playSweetly(){
          play();           //works! 'through inheritance'
          System.out.println(this+" playSweetly()");

     }
}
```

```
package classic;            //different package
import plain.PlainFiddle;

class ClassicFiddle extends PlainFiddle{
     void playSweetly(){
          PlainFiddle p = new PlainFiddle();
          p.play();         //won't work!
     }                      //not'through inheritance'
}
```

After trying the above – test the three options 1, 2 and 3 below and see if they work? In each case - reason as to why they do, or do not work.

```
…
void playSweetly(){      //change to call ‘play();’ as previously
         play();
         System.out.println(this+" playSweetly()");
}

public static void main(String[] args){
    /*1.
    ClassicFiddle c = new ClassicFiddle();
    c.playSweetly();
    //*/
    /*2.
    PlainFiddle p1 = new PlainFiddle();
    p1.play();          //does this work? why/why not?
    //*/
    /*3.
    PlainFiddle p2 = new ClassicFiddle();
    p2.play();          //does this work? why/why not?
    //*/
}
…
```

Scenario 3: More Complex Protected Access

Analyse the following code until you can clearly explain to someone else why the call to `appleType.taste()` works or does not work.

```
package fruits;
public abstract class Fruit{
     protected abstract void taste();

}
```

```
package fruits.apples;
//import fruits.Fruit;
public class Apple extends fruits.Fruit{
     String type;

     public Apple(String aType)
     {
          type = aType;
     }
     public void taste(){
          System.out.println("yum");
     }
}
```

```
package drinks;
import fruits.Fruit;
import fruits.apples.Apple;
class Cider{

  Fruit appleType;

  Cider(){
     appleType = new Apple("coxes");
     appleType.taste(); //does this work?
  }

}
```