Example: Person-Student

Consider the following two classes.

Person Class

```
public class Person {

   private String name;

   public Person(String name) {
      this.name = name;
      }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   @Override
   public String toString() {
      return name;
   }
}
```

Student Class

```
public class Student extends Person {

   private int studentNumber;

   public Student(int studentNumber,
                  String name) {
      super(name);
      this.studentNumber =
                     studentNumber;
   }

   public int getStudentNumber() {
      return studentNumber;
   }

   @Override
   public String toString() {
      return studentNumber + " - " +
                     super.toString();
   }
}
```

And now consider the following code in the same package as the above two classes.

	Test code	***Output***
1	`Person p = new Person("Chris");`	
2	`System.out.println(p);`	`Chris`
3	`p = new Student(20250123, "Betty");`	
4	`System.out.println(p);`	`2020123 - Betty`
5	`Student s = new Student(20250124,`	
6	`"Aurthur");`	
7	`System.out.println(s);`	`20250123 – Aurthur`
8	`System.out.println(s.getName());`	`Aurthur`
9	`System.out.println(s.getStudentNumber());`	`20250123`
10	`System.out.println(p.getName());`	`Betty`
11	`System.out.println(p.getStudentNumber());`	`*** ERROR ***`

The final line of the test code, line `11`, results in an error – but why? The output of the error will be something similar to:

```
The method getStudentNumber() is undefined for the type Person
```

This is a ***compile-time error***. During compilation, the compiler checks methods that are called on variables. Even though the object is of type `Student`, the reference variable, `p` that points to it is of type `Person`. The `Person` class does not have a `getStudentNumber` method, so the compiler flags it as an error and the code fails to compile – meaning it will not even reach the run-time phase.