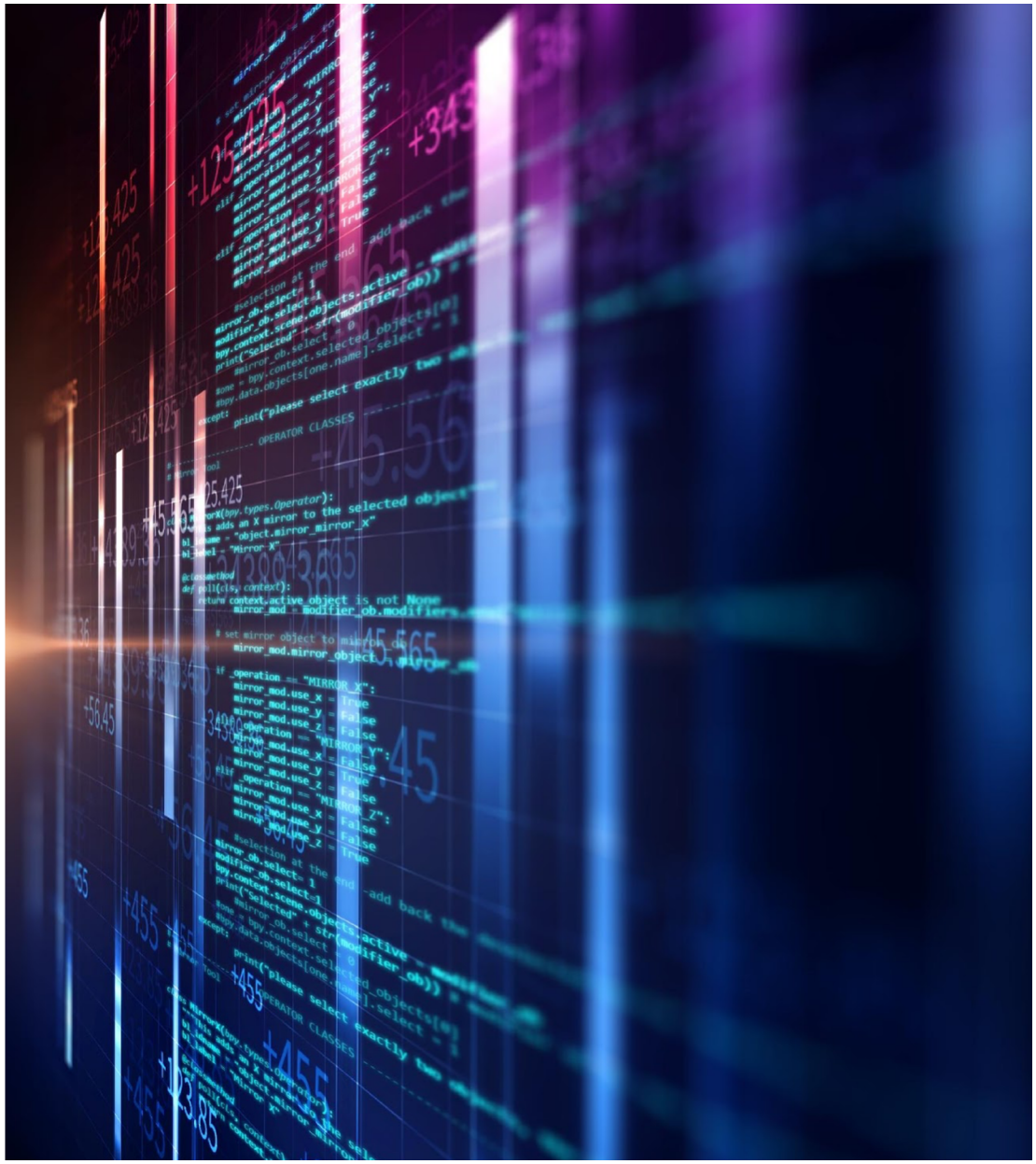
Access Modifiers

Brandon Krakowsky

Penn Engineering



Access Modifiers

Access modifiers in Java help to restrict the scope of a class, constructor, variable, or method

- There are four types of access modifiers available
  - Public: Specified using the keyword *public*
    - The public access modifier has the widest scope among all other access modifiers
    - Allows members to be accessible from any where in the program
  - Protected: Specified using the keyword *protected*
    - Allows members to be accessible within same package or sub classes in different package
  - Default (package-private): When no access modifier is specified
    - Allows members to be accessible only within the same package
  - Private: Specified using the keyword *private*
    - Allows members to be accessible only within the class in which they are declared
    - Any other class of same package will not be able to access these members

Access Modifiers

- Here's a Person class with different access modifiers used throughout

```
package people;

public class Person { //public class accessible anywhere

    public String name; //public variable accessible anywhere

    //protected variable accessible anywhere in this package (people)
    //or in sub-classes in other packages
    protected char gender;

    int age; //default variable accessible anywhere in this package (people)

    private String phone; //private variable only accessible within this class

    public String getPhone() { //public method accessible anywhere
        return this.phone; //getting private variable
    }

    public void setPhone(String phone) { //public method accessible anywhere
        this.phone = phone; //setting private variable
    }
}
```

Access Modifiers

	Class	Package	Subclass	World
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
Default	Y	Y	N	N
Private	Y	N	N	N

Access Modifiers

Which access modifier should I use?

- Start with the *most restrictive access level* that makes sense for a particular member
  - Use *private*, unless you have a good reason not to
- Avoid *public* fields, whenever possible
- If you plan on extending a class from within the package, use *default* (no keyword) for methods and fields you wish to access in the subclass
  - You can also use *default* if you plan on unit testing a method
- If you need to extend a class from outside of the package, use *protected*

Getters & Setters

Encapsulation

In Java, *getters* and *setters* are conventional methods that are used for retrieving and updating the value of a variable

- Typically, *getters* and *setters* provide access to *private* or *protected* variables

The process of making the fields in a class *private* and providing access to the fields via public methods is known as “encapsulation”

- If a field is *private*, it cannot be accessed by anyone outside the class, thereby hiding the field within the class, also referred to as “data hiding”
- *Private* variables also help prevent people from depending on certain parts of your code
  - You want users of your classes to not care how you implemented it, but rather just use the implementation through well-defined methods (e.g. getters and setters)
  - If no one is depending on your implementation, you can change it whenever you want

Encapsulation

- The following code is an example of a simple class with a *private* variable and a *public* getter and setter method

```
public class SimpleGetterAndSetter {

    private int number; //can't access outside of the class

    //provides public access outside of the class
    public int getNumber() {
        return this.number; //gets private number
    }

    //provides public access outside of the class
    public void setNumber(int num) {
        this.number = num; //sets private number
    }
}
```