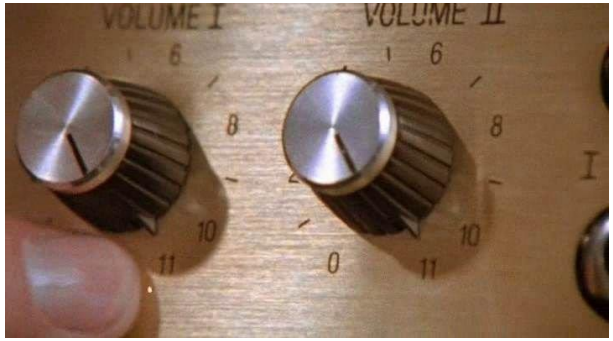

He died in a bizzare gardening accident


VOLUME I
VOLUME II
6
8
10
11
0


It was really one of those things, you know. The authorities said, you know, well...best to leave it unsolved, really.

Module 1-13

Managing Inheritance

Objectives

- Should be able to define and use `abstract` in the context of a class and a method
- Should be able to define and use `final` in the context of a class and a method
- Should understand what a design pattern is and how to research them
- Should be able to explain the differences between public, private, and protected access
- Should understand that many keywords in Java are not for security, but for design and letting other developers know how to use your code

If else if chain vs. switch statement

```
…
System.out.println(“Movie ticket prices: “);
System.out.println(“1. Adult - $14.00”);
System.out.println(“2. Child - $8.00”);
 System.out.println(“3. Senior - $11.00”);
System.out.print(“Enter choice: “);
int choice = Integer.parseInt(input.nextLine());
       if (choice == 1) {
          total = quantity * 14;
       } else if (choice == 2) {
           total = quantity * 8;
       }  else if (choice == 3) {
            total = quantity * 11;
       }  else {
            System.out.println(“Invalid entry”);
       }

  …
```

```
…
System.out.println(“Movie ticket prices: “);
System.out.println(“1. Adult - $14.00”);
System.out.println(“2. Child - $8.00”);
 System.out.println(“3. Senior - $11.00”);
System.out.print(“Enter choice: “);
int choice = Integer.parseInt(input.nextLine());
switch (choice) {
       case 1:
            total = quantity * 14;
            break;
       case 2:
            total = quantity * 8;
            break;
       case 3:
            total = quantity * 11;
            break;
       default:
            System.out.println(“Invalid entry”);
  }

  …
```

Making Animals Sleep

Abstract Classes

Abstract Classes combine some of the features we've seen in interfaces along with inheriting from a concrete class.

- Abstract methods can be extended by concrete classes.
- Abstract classes can have abstract methods
- Abstract classes can have concrete methods
- Abstract classes can have constructors
- Abstract classes, like Interfaces, cannot be instantiated

Abstract Classes : Declaration

We use the following pattern to declare abstract classes.

- The abstract class itself:

  public **abstract** class **<<Name of the Abstract Class>>** {...}

- The child class that inherits from the abstract class:

  public class **<<Name of Child Class>>** **extends** **<<Name of Abstract Class>>**

Abstract Classes Example

```
package te.mobility;

public abstract class Vehicle {

     private int numberOfWheels;
     private double tankCapacity;
     private double fuelLeft;

     public Vehicle(int numberOfWheels) {
        this.numberOfWheels = numberOfWheels;
     }
     public double getTankCapacity() {
        return tankCapacity;
     }

     public abstract Double calculateFuelPercentage();


     public double getFuelLeft() {
       return fuelLeft;
     }
}
```

We need to implement the constructor

We need to implement the abstract method

extends, not implement, is used.

```
package te.mobility;

public class Car extends Vehicle {

     public Car(int numberOfWheels) {
        super(numberOfWheels);
     }

     @Override
     public Double calculateFuelPercentage() {

        return super.getFuelLeft() /
           super.getTankCapacity() * 100;
     }

}
```

Also note how we are able to call concrete methods within the Vehicle abstract class

Abstract Classes: final keyword

Declaring methods as **`final`** prevent them from being overriden by a child class.

```
package te.mobility;

public abstract class Vehicle {
...

    public final void refuelCar() {
       this.fuelLeft = tankCapacity;
...
}
```

```
package te.mobility;

public class Car extends Vehicle
{

    @Override
    public void refuelCar() {

    }

}
```

This override will cause an error, as the method is marked as final.

Multiple Inheritance

- Java does not allow multiple inheritance of concrete classes or abstract classes. The following **is not allowed**:

  public class Car **extends Vehicle, MotorVehicles** {...}

Where Vehicle and MotorVehicles are classes or abstract classes

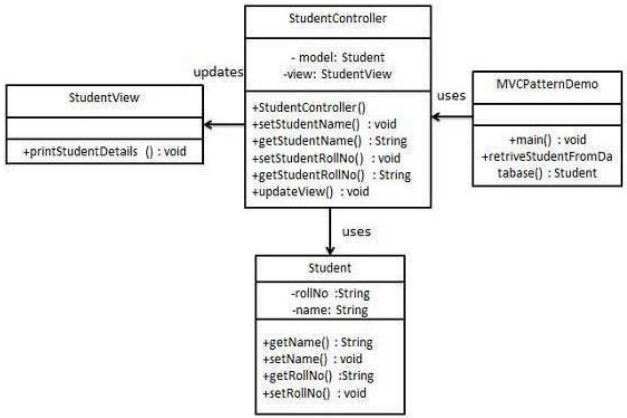
- Java does allow for the implementation of multiple interfaces:

  public class Car **implements IVehicle, IMotorVehicle** {...}

Where IVehicle and IMotorVehicle are interfaces

Design Patterns

- Represent best practices used by experienced object-oriented software developers.
- Solutions to general problems that software developers faced during software development..



https://www.tutorialspoint.com/design_pattern/index.htm

ABSTRACT CLASSES VS INTERFACES

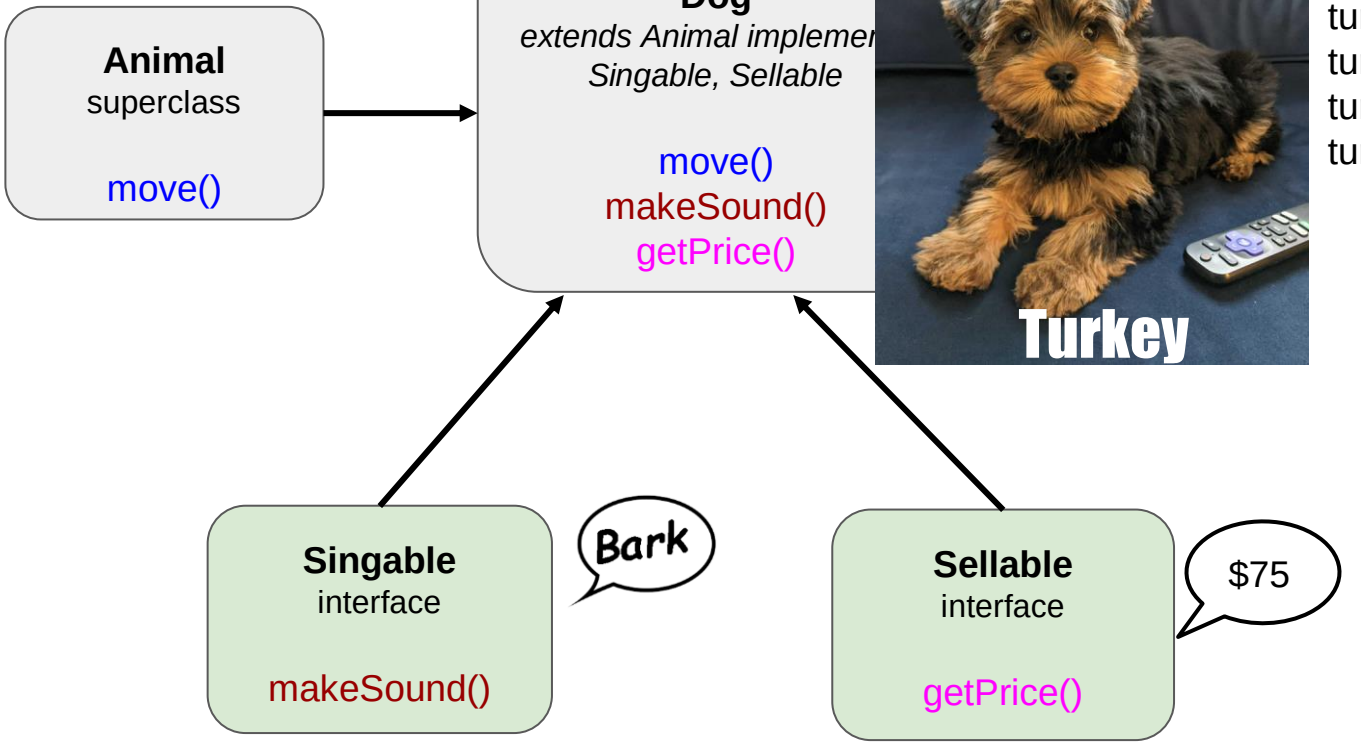
ABSTRACT CLASS

- Defines methods & properties
- Can contain method bodies
- Can contain properties
- Cannot be instantiated
- Are inherited

INTERFACES

- Defines methods & properties
- No method bodies
- No properties
- Cannot be instantiated
- Are implemented

Object instanceof



Dog turkey = new Dog();

turkey *instanceof* Animal
turkey *instanceof* Dog
turkey *instanceof* Singable
turkey *instanceof* Sellable

An Object is an instanceof a data type if it can be safely cast to that data type.

instanceof operator

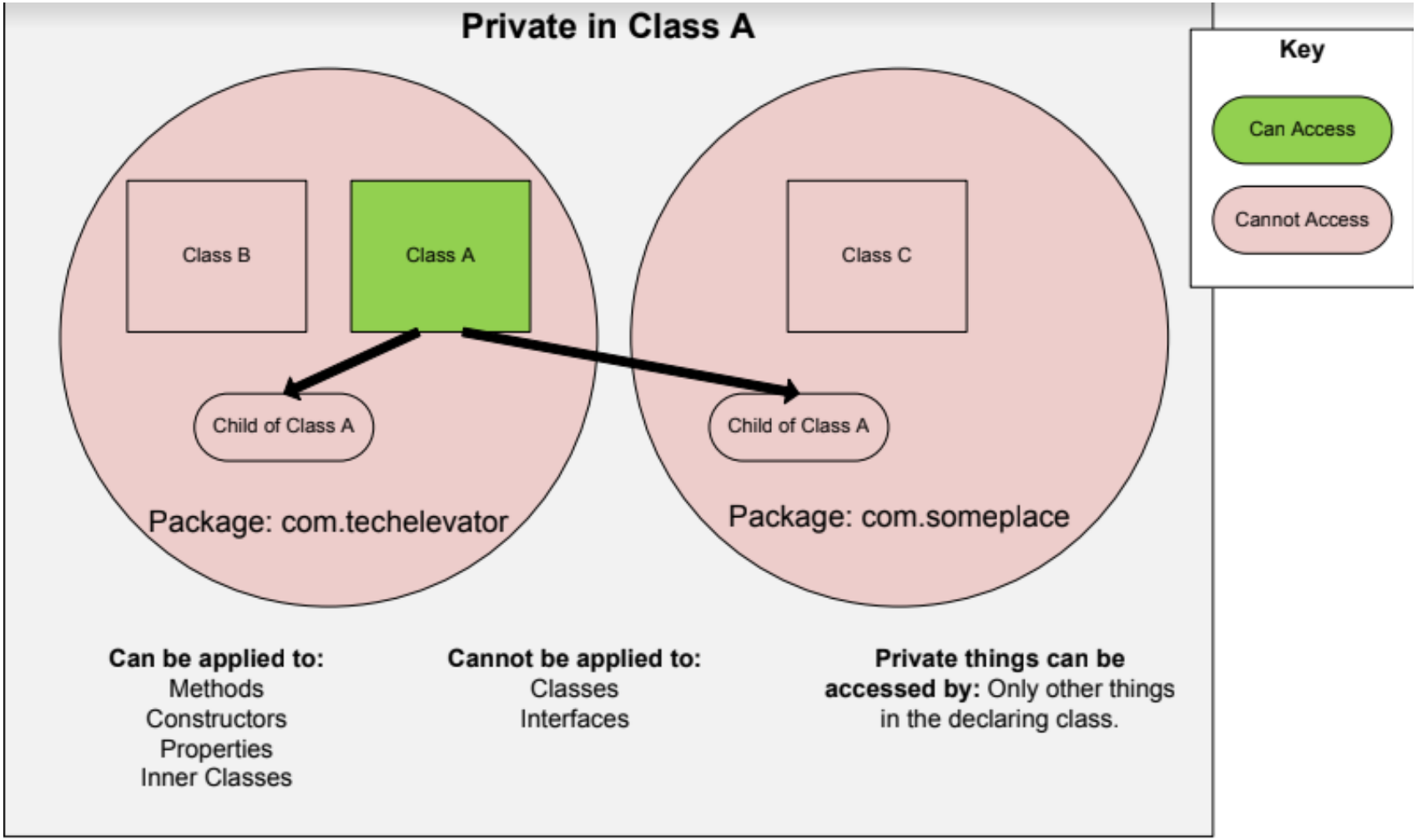
Since *downcasting* (treating a parent class like the child) can only be done if the object is already internally the type it is being cast to, there is a boolean operator, ***instanceof***, that can check if the object can be downcast to the subclass type.

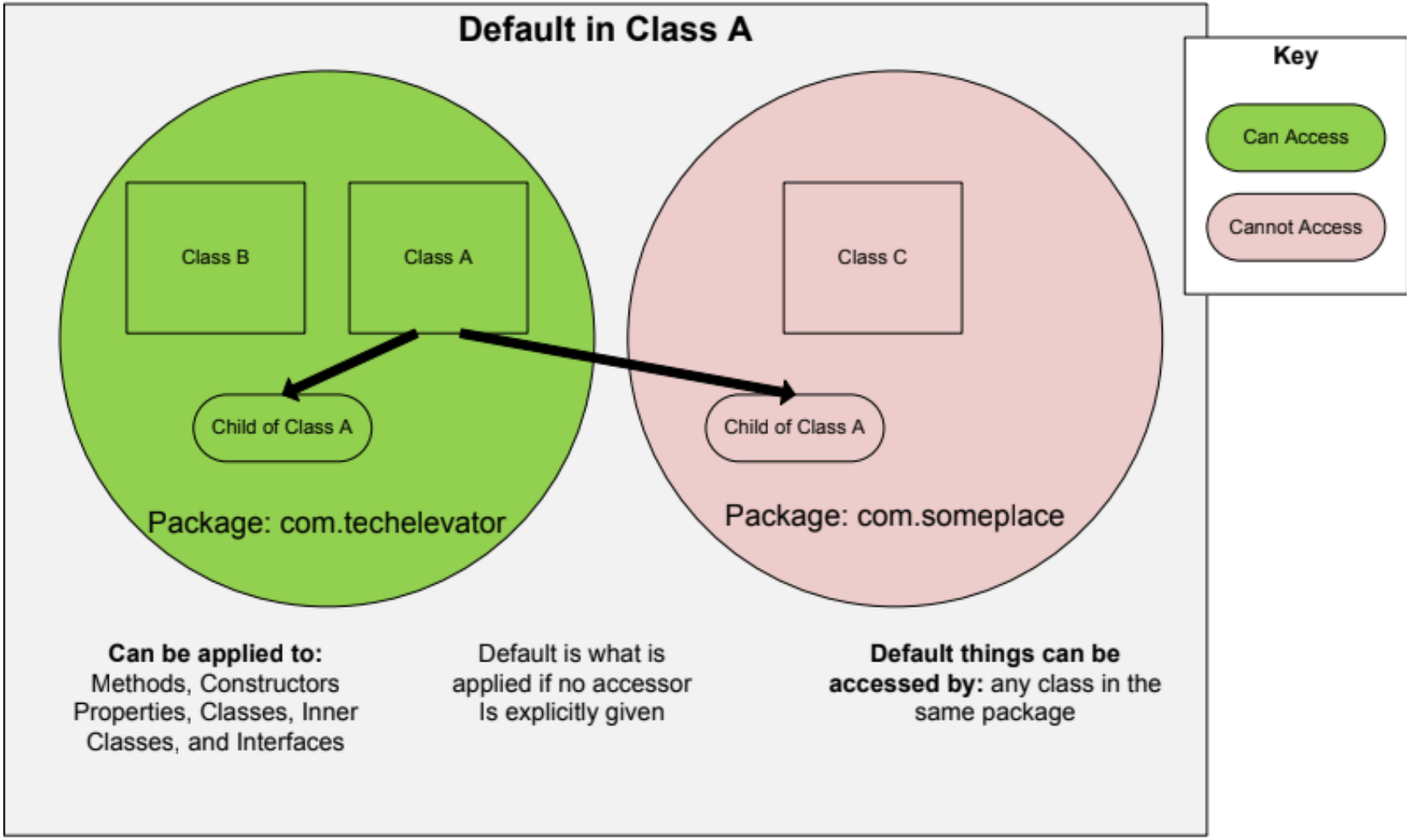
object instanceof class

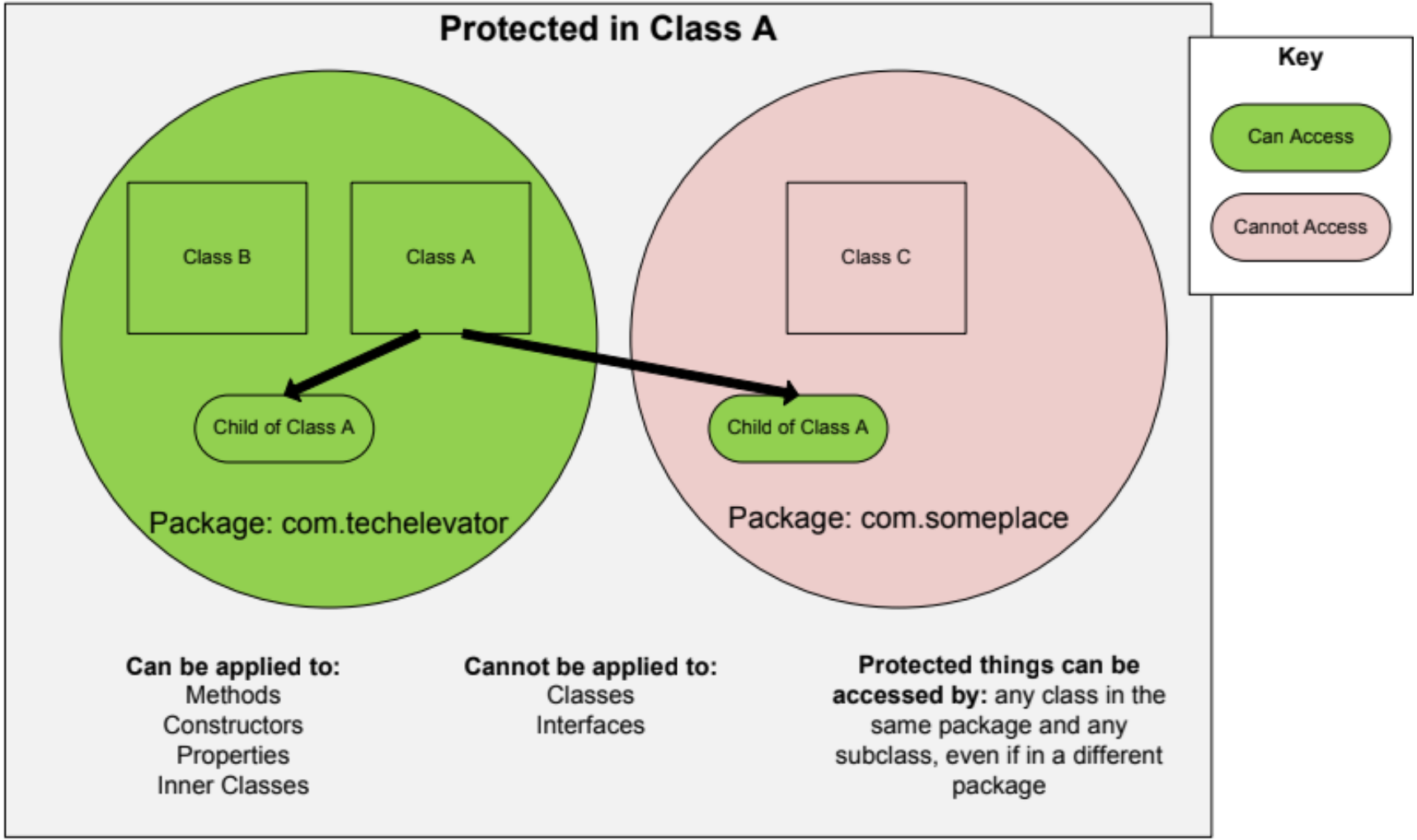
```
public void convert(Calculator calculator) {

     if (calculator instanceof ScientificCalculator) {
               ScientificCalculator sc = (ScientificCalculator) calculator;
     }

}
```

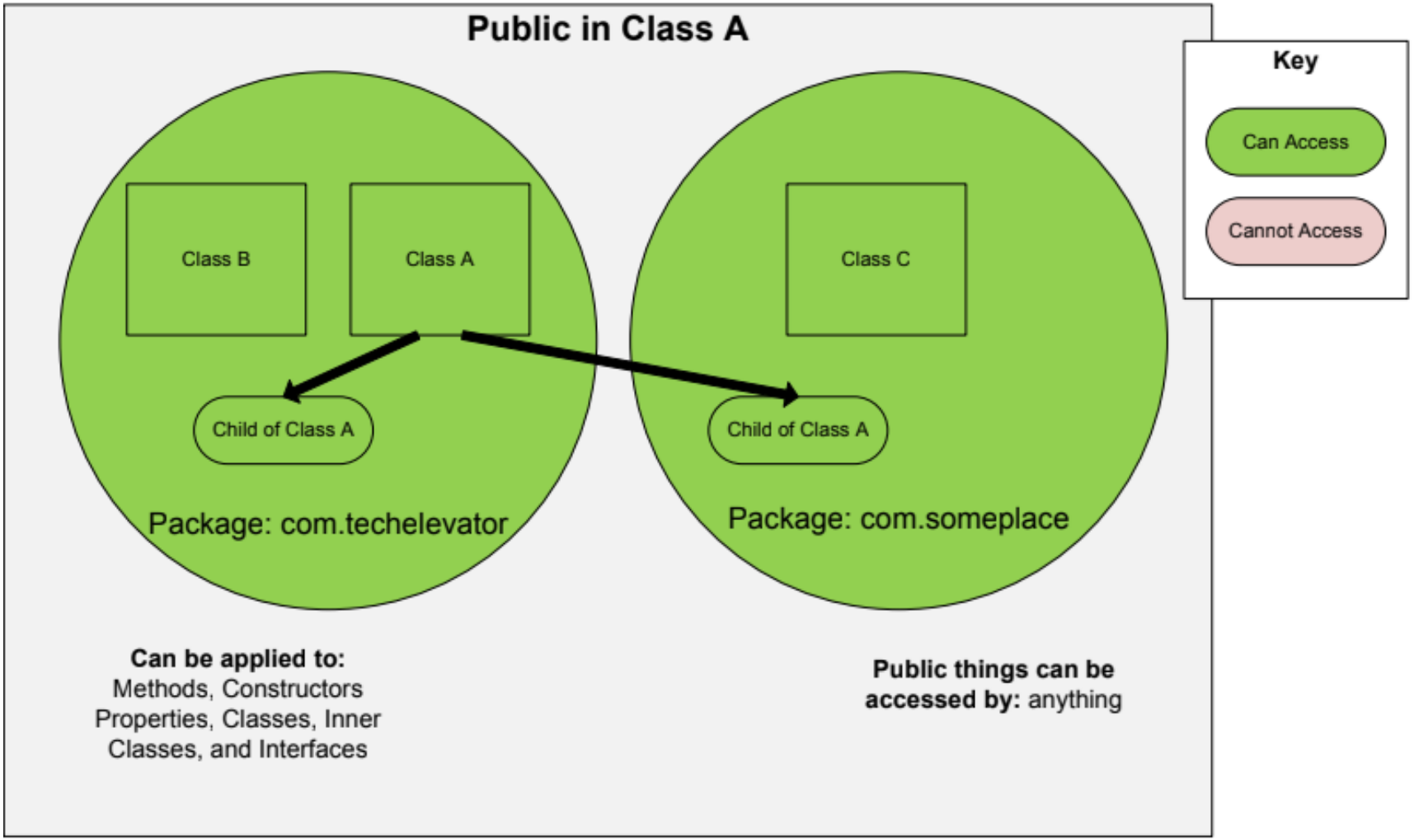
instanceof should be used when a class is being downcast to a subclass, and it is not known what type the object is internally.

instanceof never needs to be used when upcasting to a superclass, since all subclasses can always be upcast to their superclass.


Private in Class A
Key
Can Access
Cannot Access
Class B
Class A
Class C
Child of Class A
Child of Class A
Package: com.techelevator
Package: com.someplace
Can be applied to:
Methods
Constructors
Properties
Inner Classes
Cannot be applied to:
Classes
Interfaces
Private things can be accessed by: Only other things in the declaring class.


Default in Class A
Key
Can Access
Cannot Access
Class B
Class A
Class C
Child of Class A
Child of Class A
Package: com.techelevator
Package: com.someplace
Can be applied to: Methods, Constructors Properties, Classes, Inner Classes, and Interfaces
Default is what is applied if no accessor Is explicitly given
Default things can be accessed by: any class in the same package


Protected in Class A
Key
Can Access
Cannot Access
Class B
Class A
Class C
Child of Class A
Child of Class A
Package: com.techelevator
Package: com.someplace
Can be applied to:
Methods
Constructors
Properties
Inner Classes
Cannot be applied to:
Classes
Interfaces
Protected things can be accessed by: any class in the same package and any subclass, even if in a different package


Public in Class A
Key
Can Access
Cannot Access
Class B
Class A
Class C
Child of Class A
Child of Class A
Package: com.techelevator
Package: com.someplace
Can be applied to:
Methods, Constructors
Properties, Classes, Inner
Classes, and Interfaces
Public things can be
accessed by: anything

When to use each accessor

Access	Visibility	Reason to use it
public	Everyone	for “set in stone” methods that you want other programmers to rely on to use your object. These create the behaviors of the object, but changing their method signatures may break other code that is using your object.
protected	Subclasses	for building connections between inherited classes. It lets you have methods in a superclass that are accessible to the subclasses, but does not allow access outside the hierarchy.
default	Package	for building cohesion between related classes in the same package. should generally be avoided.
private	Class	for unstable, worker methods that may change and are only for use inside the class itself.

Class design should include how others will use your object, the methods that allow that use should be public. All other methods and variables should be private, until needed in the hierarchy or publically.

static methods

If we define a method static, our class does not need to be instantiated to use it. Instead it can be accessed from the Class itself, instead of the object. *Static methods can only access other static methods or variables.*

```
public class Rectangle() {

        private static int length;
        private static int width;

        public static getArea() {
                    return length * width;
        }
}
```

```
private static void main(String[] args)
```

In the class using the static method:

```
        Rectangle.getArea();

        Rectangle rect = new Rectangle();
        rect.getArea();
```

This seems easier, why not make everything static?

Examples

Math.abs()
Math.random()

String.join()
String.valueOf()

Double.parseDouble()
Integer.parseInt()

Objectives

- Should be able to define and use abstract in the context of a class and a method

Objectives

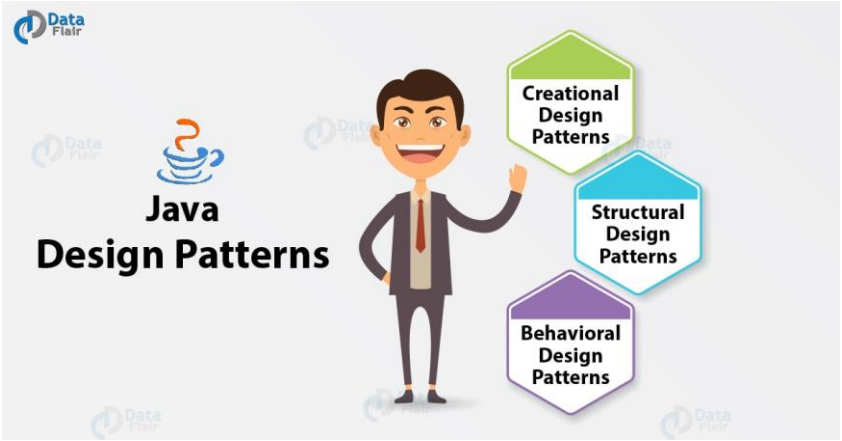
- Should be able to define and use abstract in the context of a class and a method
- Should be able to define and use final in the context of a class and a method

Objectives

- Should be able to define and use abstract in the context of a class and a method
- Should be able to define and use final in the context of a class and a method
- Should understand what a design pattern is and how to research them

https://www.javatpoint.com/design-patterns-in-java

https://cs.lmu.edu/~ray/notes/designpatterns/



Objectives

- Should be able to define and use abstract in the context of a class and a method
- Should be able to define and use final in the context of a class and a method
- Should understand what a design pattern is and how to research them
- Should be able to explain the differences between public, private, and protected access

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Objectives

- Should be able to define and use abstract in the context of a class and a method
- Should be able to define and use final in the context of a class and a method
- Should understand what a design pattern is and how to research them
- Should be able to explain the differences between public, private, and protected access
- Should understand that many keywords in Java are not for security, but for design and letting other developers know how to use your code