

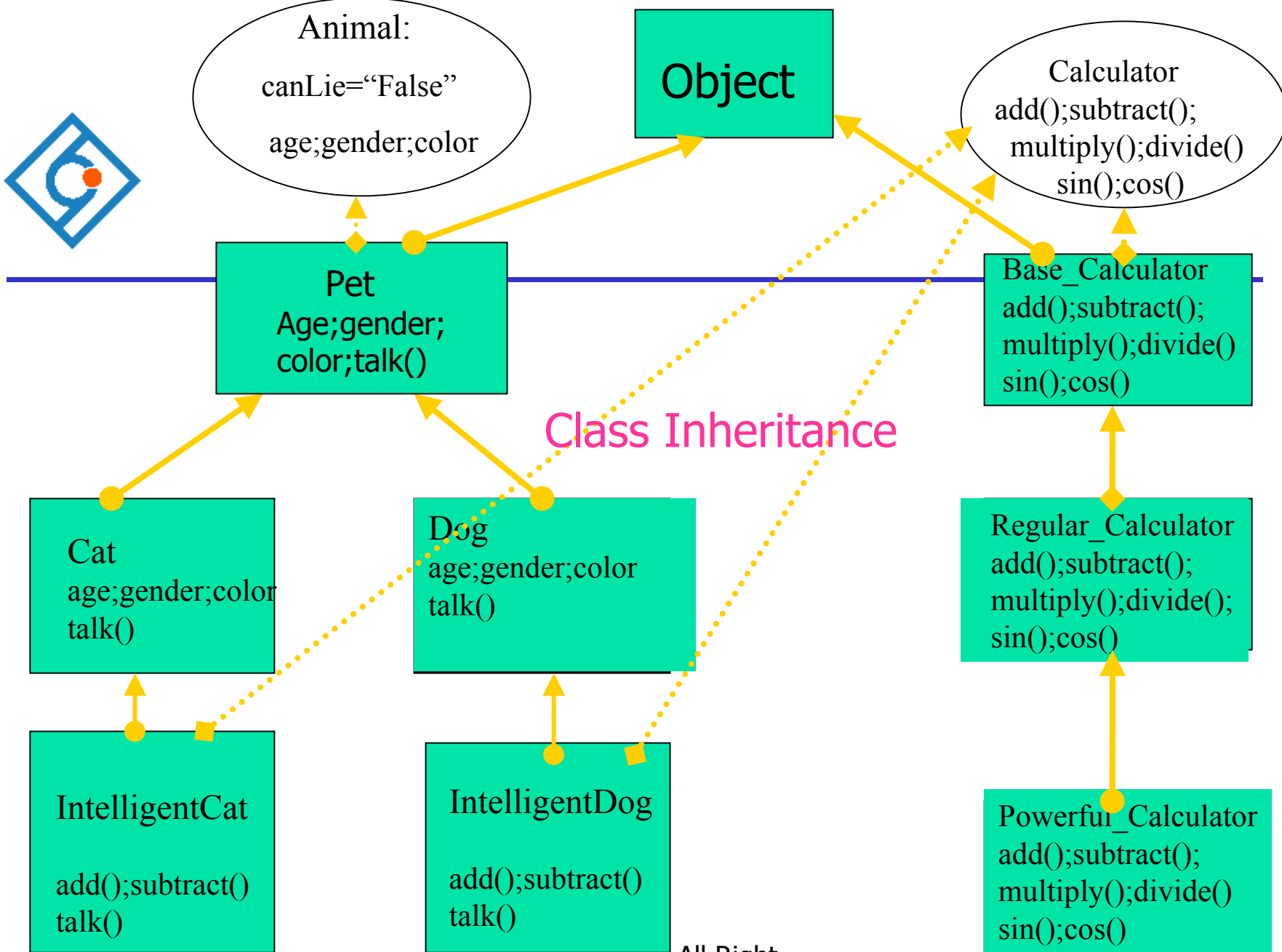


Toronto College of Technology

---

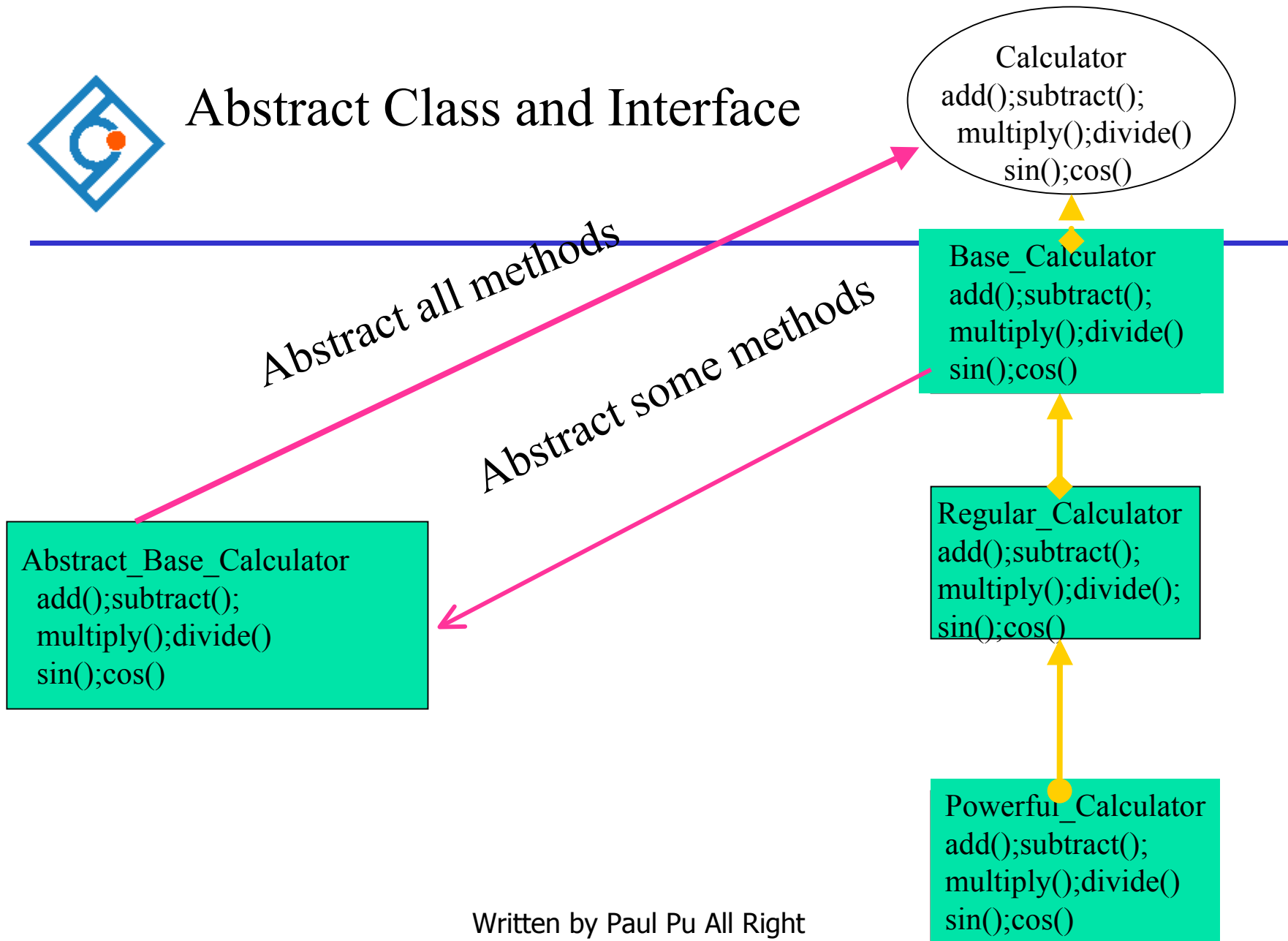
# Abstract Class and Interface

Written by Paul Pu All Right  
Reserved [www.torontocollege.com](http://www.torontocollege.com)





# Abstract Class and Interface





# Abstract Class and Interface

---

- **What Is an Abstract Class?**
- **Definition:** An Abstract Class is class which has one or more than one abstract method (methods).
- **Warning:** You can not use abstract class to create object.



# Abstract Class and Interface

---

- **What Is an Interface?**
- **Definition:** An *interface* is a named collection of method definitions (without implementations). An interface can also declare constants.



# Abstract Class and Interface

---

- **An *interface*** defines a protocol of behavior that can be implemented by **any class** anywhere in the class hierarchy. An interface defines a set of methods but does not implement them. *A class that implements the interface agrees to implement all the methods defined in the interface, thereby agreeing to certain behavior.*



# Abstract Class and Interface Difference

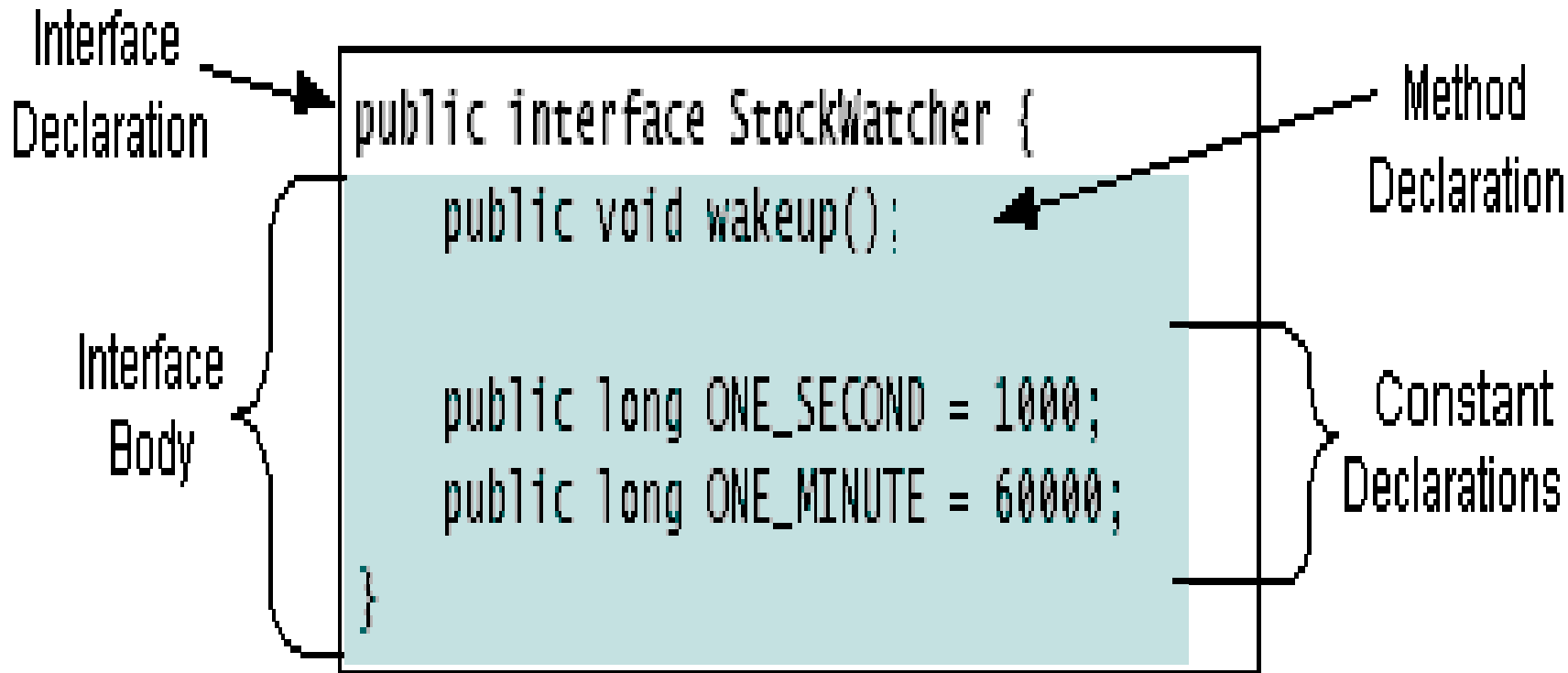
---

- Because an interface is simply a list of unimplemented, and therefore abstract, methods, you might wonder how an interface differs from an abstract class. The differences are significant.
- An interface cannot implement any methods, whereas an abstract class can.
- A class can implement many interfaces but can have only one superclass.
- An interface is not part of the class hierarchy. Unrelated classes can implement the same interface



# Defining an Interface

---





# Defining an Interface

---

## Defining an Interface

<code>public</code>	Makes this interface public.
<code>interface InterfaceName</code>	Class cannot be instantiated.
<code>Extends SuperInterfaces</code>	This interface's superinterfaces.
<pre>{     <i>InterfaceBody</i> }</pre>	



# Packages

---

- **Creating and Using Packages**
- **Definition:** A *package* is a collection of related classes and interfaces providing access protection and namespace management



# Packages

---

To make classes easier to **find and to use**, to **avoid naming conflicts**, and **to control access**, programmers bundle groups of related classes and interfaces into packages.



# Packages

---

- You should bundle these classes and the interface in a package for several reasons:
- You and other programmers can easily determine that these classes and interfaces are related.
- You and other programmers know where to find classes and interfaces that provide graphics-related functions.



# Packages

---

- The names of your classes won't conflict with class names in other packages, because the package creates a new namespace.
- You can allow classes within the package to have unrestricted access to one another yet still restrict access for classes outside the package



# Packages

---

- **By Convention:** Companies use their reversed Internet domain name in their package names, like this: `com.company.package`. Some companies now choose to drop the first element `com.` in this example from their package names. Name collisions that occur within a single company need to be handled by convention within that company, perhaps by including the region or the project name after the company name, for example, `com.company.region.package`.