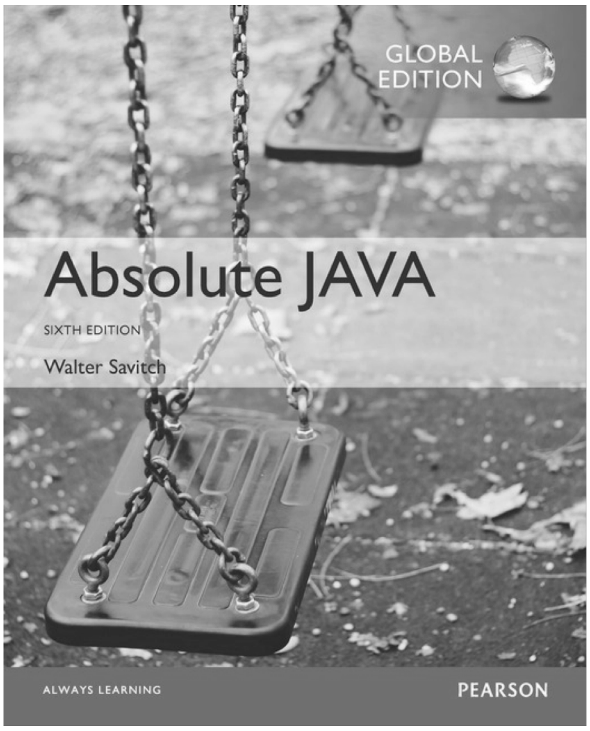

# Chapter 8

Abstract Classes

Slides prepared by Rose Williams, *Binghamton University*

Kenrick Mock, *University of Alaska Anchorage*

## Introduction to Abstract Classes

- In Chapter 7, the **`Employee`** base class and two of its derived classes, **`HourlyEmployee`** and **`SalariedEmployee`** were defined
- The following method is added to the **`Employee`** class
  - It compares employees to to see if they have the same pay:

```
public boolean samePay(Employee other)
{
  return(this.getPay() == other.getPay());
}
```

## Introduction to Abstract Classes

- There are several problems with this method:
  - The `getPay` method is invoked in the `samePay` method
  - There are `getPay` methods in each of the derived classes
  - There is no `getPay` method in the `Employee` class, nor is there any way to define it reasonably without knowing whether the employee is hourly or salaried

## Introduction to Abstract Classes

- The ideal situation would be if there were a way to
  - Postpone the definition of a `getPay` method until the type of the employee were known (i.e., in the derived classes)
  - Leave some kind of note in the `Employee` class to indicate that it was accounted for
- Surprisingly, Java allows this using abstract classes and methods

## Introduction to Abstract Classes

- In order to postpone the definition of a method, Java allows an *abstract method* to be declared
  - An abstract method has a heading, but no method body
  - The body of the method is defined in the derived classes
- The class that contains an abstract method is called an *abstract class*

## Abstract Method

- An abstract method is like a placeholder for a method that will be fully defined in a descendent class
- It has a complete method heading, to which has been added the modifier **`abstract`**
- It cannot be private
- It has no method body, and ends with a semicolon in place of its body

```
public abstract double getPay();
public abstract void doIt(int count);
```

## Abstract Class

- A class that has at least one abstract method is called an *abstract class*
  - An abstract class must have the modifier **`abstract`** included in its class heading:

```
public abstract class Employee
{
  private instanceVariables;
  . . .
  public abstract double getPay();
  . . .
}
```

## Abstract Class

- An abstract class can have any number of abstract and/or fully defined methods
- If a derived class of an abstract class adds to or does not define all of the abstract methods, then it is abstract also, and must add **`abstract`** to its modifier

- A class that has no abstract methods is called a *concrete class*

## Pitfall: You Cannot Create Instances of an Abstract Class

- An abstract class can only be used to derive more specialized classes
  - While it may be useful to discuss employees in general, in reality an employee must be a salaried worker or an hourly worker
- An abstract class constructor cannot be used to create an object of the abstract class
  - However, a derived class constructor will include an invocation of the abstract class constructor in the form of **`super`**

## Tip: An Abstract Class Is a Type

- Although an object of an abstract class cannot be created, it is perfectly fine to have a parameter of an abstract class type
  - This makes it possible to plug in an object of any of its descendent classes
- It is also fine to use a variable of an abstract class type, as long is it names objects of its concrete descendent classes only