Week 3: Topics

- Method Overloading
- Instance and Static Variables
- Interfaces

Method Overloading in Java

- Methods of the **same** name can be declared in the same class

  As long as they have different sets of parameters

  - Known as *Method Overloading*

- Compiler selects the appropriate method to call by examining

  The Number, Types and Order of the arguments in the call

Method Overloading in Java (cont.)

- Used to create several methods that perform

  The same or similar tasks on different types or different numbers of arguments

Without Method Overloading

```
int add2(int x, int y)
{
    return(x+y);
}
int add3(int x, int y,int z)
{
    return(x+y+z);
}
int add4(int w, int x,int y, int z)
{
    return(w+x+y+z);
}
```

With Method Overloading

```
int add(int x, int y)
{
    return(x+y);
}
int add(int x, int y,int z)
{
    return(x+y+z);
}
int add(int w, int x,int y, int z)
{
    return(w+x+y+z);
}
```

Problem:

Create a class called `MethodOverload` and define a method called `square` that takes an `int` as a parameter and returns the square of the input as an `int`.

Overload the `square` method such that it takes a `double` as an input and also returns a `double`.

Create a `main()` method to call each of the above methods and print their return values.

Blackboard: Week3/MethodOverload.java

A quick tutorial:

Blackboard: Week3/Numbers.java

Method Overloading in Java

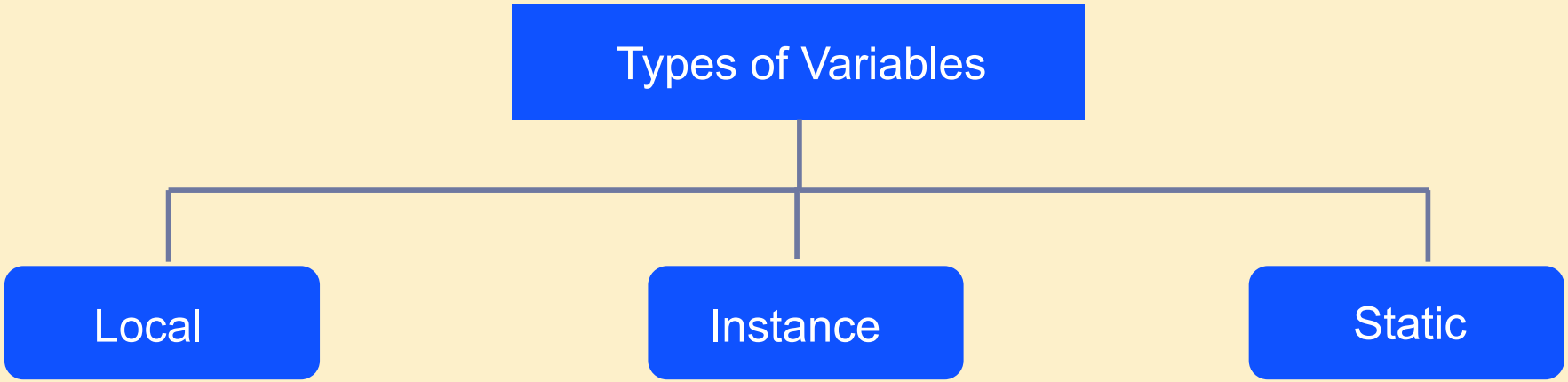
- Compiler distinguishes overloaded methods by their *signatures*

  A combination of the method's name and the number, types and order of its parameters

- Method calls cannot be distinguished

  By their return type alone

Types of Variables



Local Variables

- Declared in methods, constructors or blocks
- Created when methods, constructors or blocks are entered and destroyed once they are exited

  Do not have access modifiers
- Visible only within the declared methods, constructors or blocks

  Do not have default values

A quick tutorial:

Blackboard: Week3/Test.java

Instance Variables

- Declared in a class but outside of methods, constructors or blocks

  Have default values

- Have access modifiers

  And are usually declared as private

Blackboard: Week3/Account.java

Static Variables

- Declared with the `static` keyword in a class outside of methods, constructors or blocks

  Only one copy of each static variable per class, irrespective of the number of objects created from it

- Created when the program starts and are destroyed when the program ends
  - Most `static` variables have `public` level access – why?
- Have default values
- Accessed by `ClassName.variableName`

Blackboard: Week3/VariableDemo.java

Java Interface

- Objects describe their interaction with the outside world through the methods that they expose
- Methods form the object's **interface**
  - An interface is a description of the actions that an object can do
- A Java interface is a bit like a class, except that it can only contain *method signatures* and *fields*

Cannot contain an implementation of the methods

Java Interface Example

```
public interface NameOfInterface
{
   // Any number of final, static fields
   // Any number of abstract method declarations
}

public interface Car
{
   void changeGear(int newValue);
   void speedUp(int increment);
   void applyBrake(int decrement);
}
```

- Methods need to be implemented by some class

Blackboard: Week3/Hatchback/Hatchback.java

Java Interface Implementation

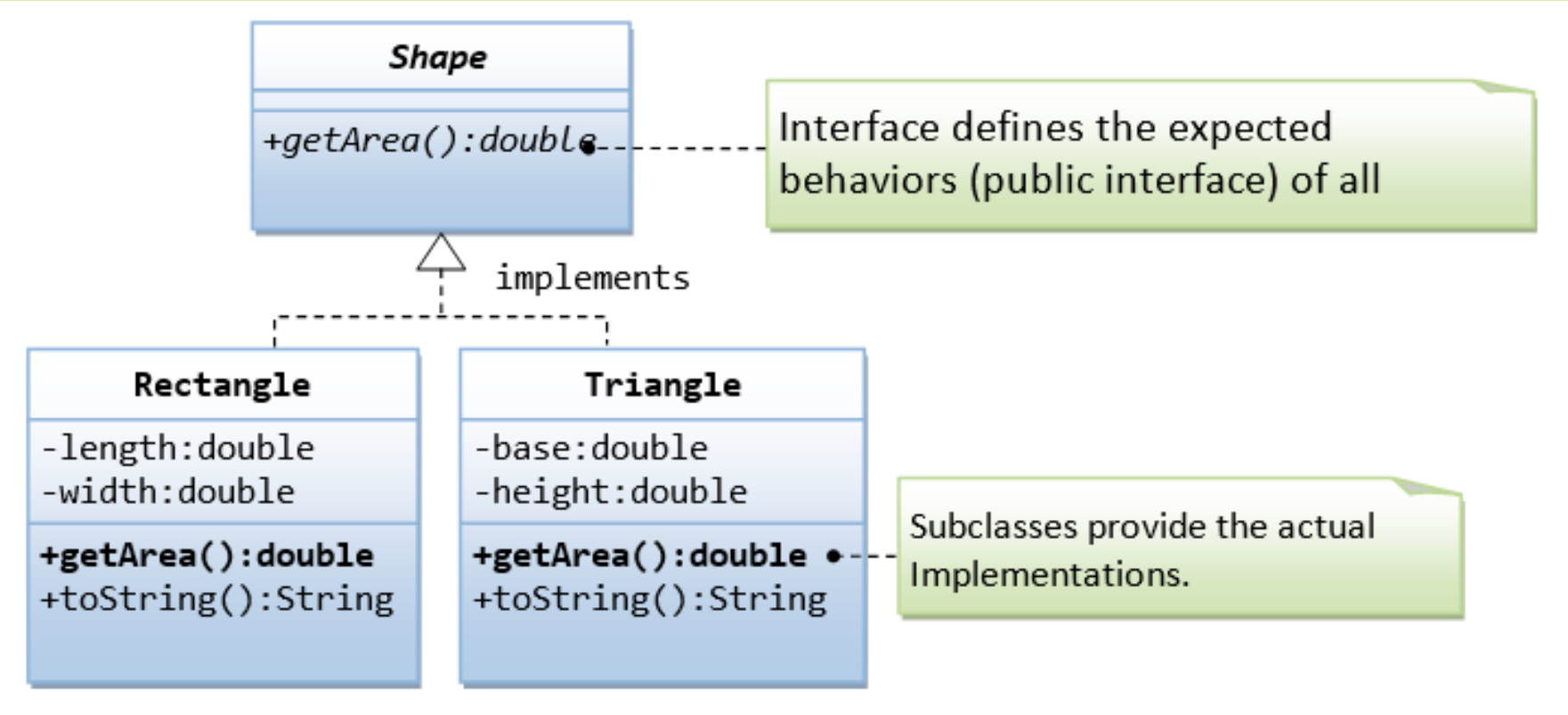
- A class that implements an interface

  Must implement all the methods declared in the interface

- The methods must have the exact same signature

  Name + parameters as declared in the interface

A quick tutorial:



Define interface `Shape` and implement its abstract method in the `Rectangle` class.

Blackboard: Week3/Interface

Collections in Java

- The Java *collections framework* is a set of classes and interfaces that implement commonly reusable collection data structures

- Operations that you perform on data such as searching, sorting, insertion, deletion etc.
  - Can be performed by Java collections

- You have used arrays to store sequence of objects
  - However arrays do not automatically change their size at execution time to accommodate additional elements

The collection class ArrayList<T> provides a convenient solution to this problem

Methods for `ArrayList`

Method	Description
`add`	Adds an element to the end of the `ArrayList`
`clear`	Removes all of the elements from an `ArrayList`
`get`	Returns the element at the specified index
……	……

Blackboard: Week3/ArrayListCollection.java

Blackboard: Week3/ArrayListCollection.java