WSA | Forward looking IT finishing school

Java Programming Language SE – 6

Module 6 : Class Design

ORACLE

Certified Professional

Java SE 6 Programmer

www.webstackacademy.com

Objectives

- Define inheritance, polymorphism, overloading, overriding, and virtual method invocation
- Use the access modifiers protected and the default

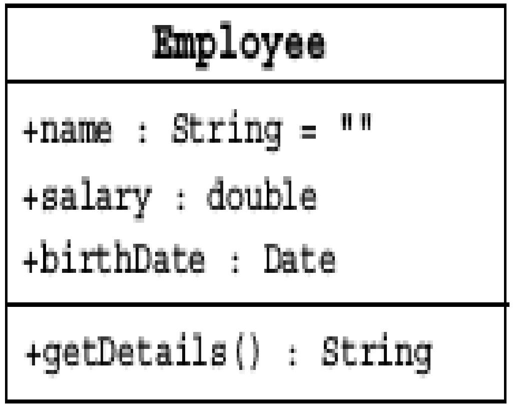
  (package-friendly)
- Describe the concepts of constructor and method overloading
- Describe the complete object construction and initialization operation

Relevance

- How does the Java programming language support object inheritance?

Subclassing

The Employee class is shown here.

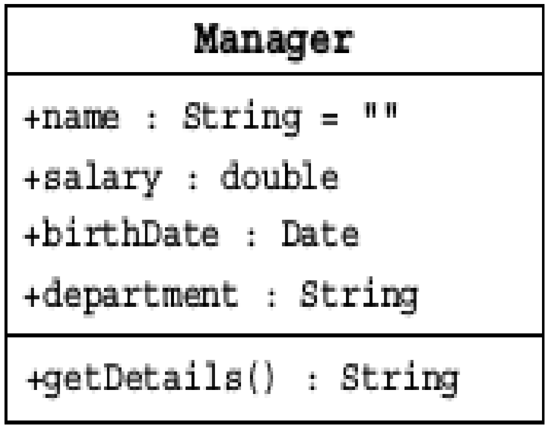


```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```

Subclassing

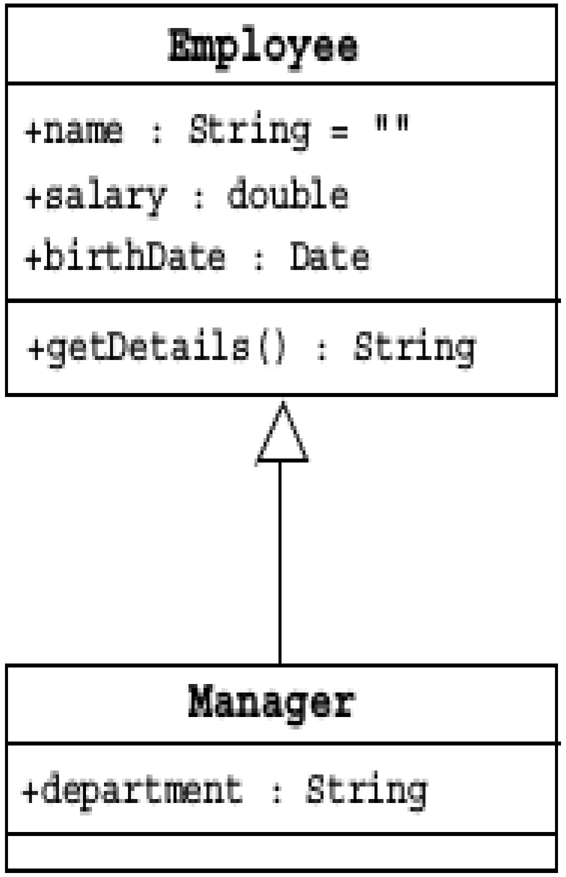
The Manager class is shown here.



```
public class Manager {
  public String name = "";
  public double salary;
  public Date birthDate;
  public String department;

  public String getDetails() {...}
}
```

Employee and Manager Using Inheritance



```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```

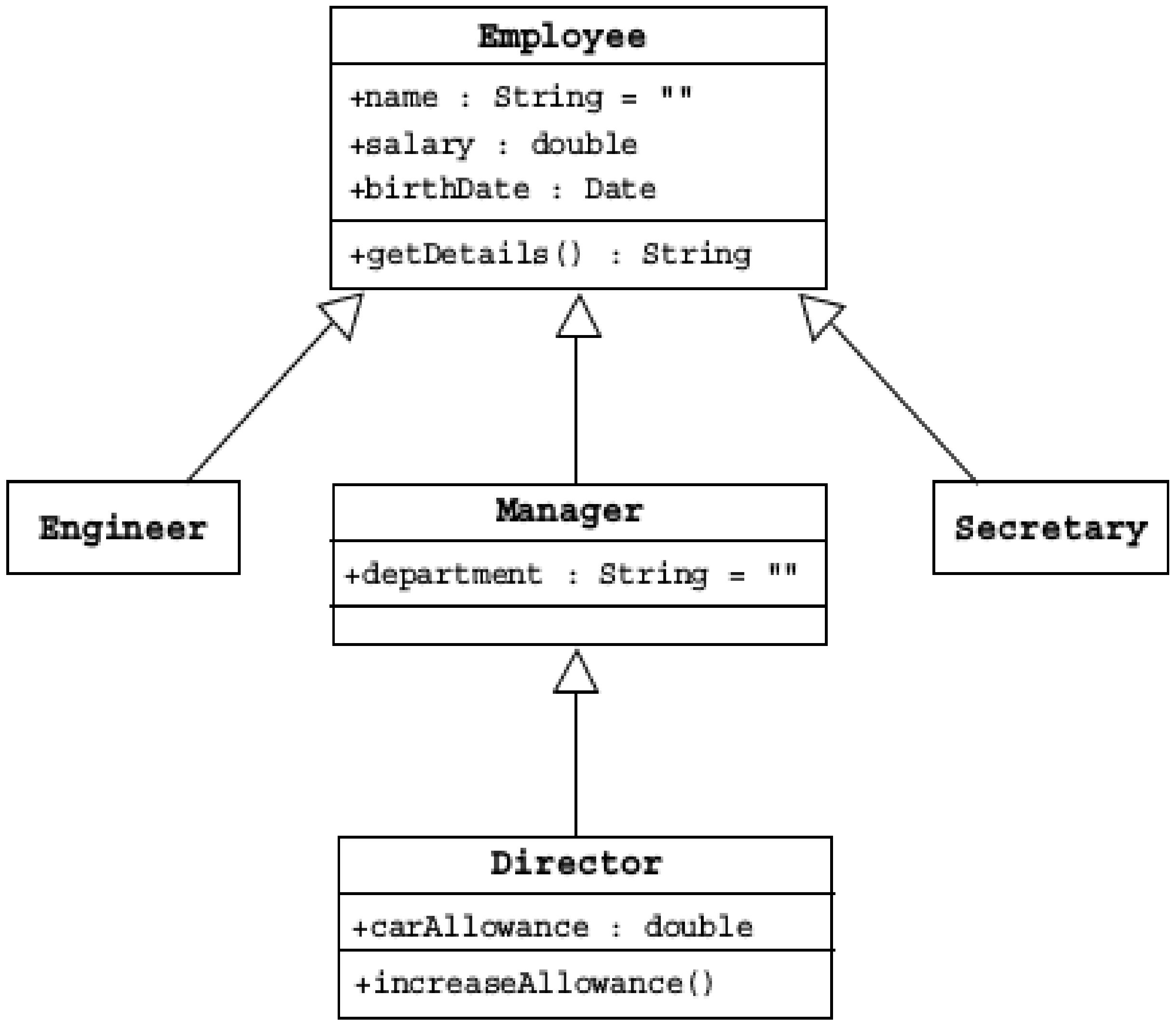
```
public class Manager extends Employee {
  public String department;
}
```

Single Inheritance

- When a class inherits from only one class, it is called single inheritance.
- Interfaces provide the benefits of multiple inheritance without drawbacks.
- Syntax of a Java class is as follows:

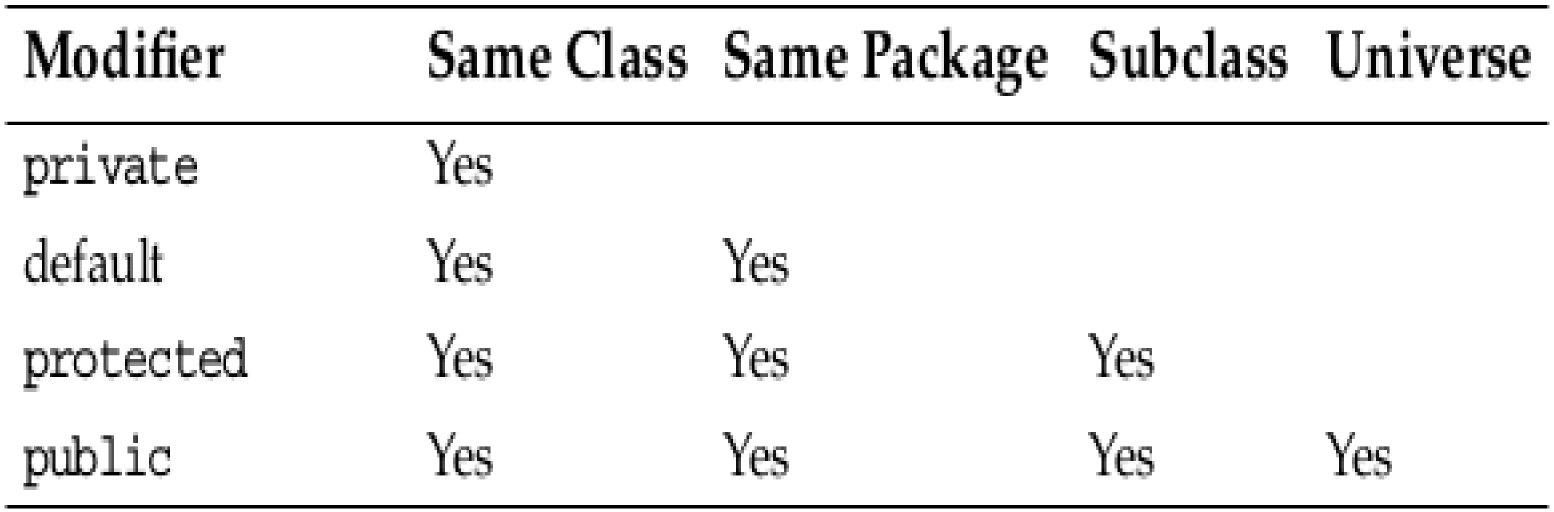
```
<modifier> class <name> [extends <superclass>] {
<declaration>*
}
```

Single Inheritance



Access Control

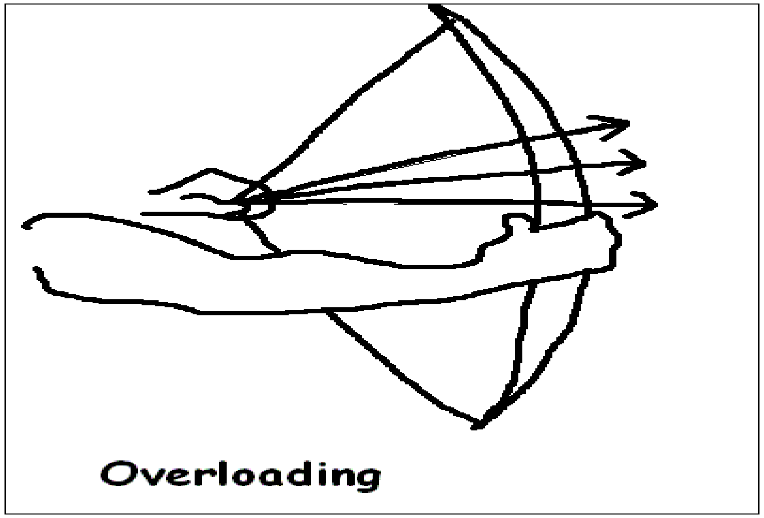
- Access modifiers on class member declarations are listed here.

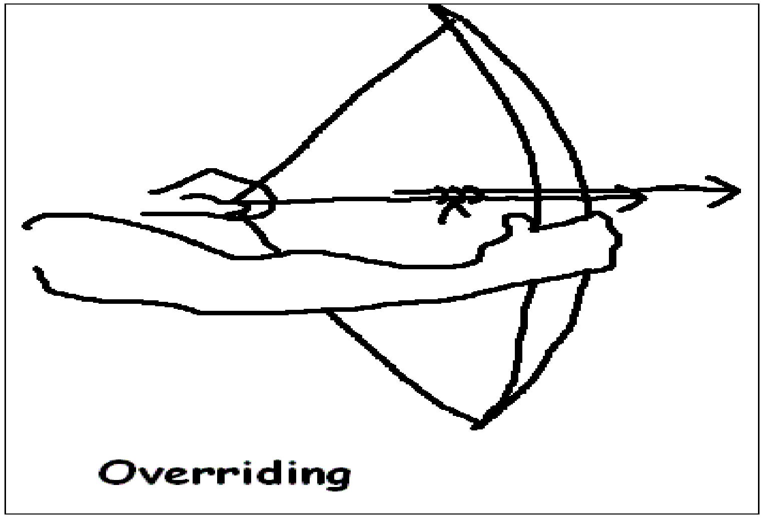


Modifier	Same Class	Same Package	Subclass	Universe
private	Yes			
default	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

Overriding Methods

- A subclass can modify behavior inherited from a parent class.
- A subclass can create a method with different functionality than the parent's method but with the same:
  - Name
  - Return type
  - Argument list





Overriding Methods

```
public class Employee {
protected String name;
protected double salary;
protected Date birthDate;
Public String getDetails(){
Return “Name:”+name+”\n”+”Salary:”+salary;
}
}
```

Overriding Methods

```
public class Manager extends Employee {
protected String department;
public String getDetails() {
return “Name: “ + name + “\n” +
“Salary: “ + salary +”\n” +”Manager of:”+ department;
}
}
```

Rules of Method Overriding

Overridden methods can't be less accessible.

```
public class Parent {
public void doSomething() {}
}
public class Child extends Parent {
private void doSomething() {} // illegal
}
```

Invoking Overridden Methods

A subclass method may invoke a superclass method using the super keyword:

- The keyword super is used in a class to refer to its superclass.
- The keyword super is used to refer to the members of superclass, both data attributes and methods.
- Behavior invoked does not have to be in the superclass; it can be further up in the hierarchy.

Invoking Overridden Methods

```
public class Employee {
protected String name;
protected double salary;
protected Date birthDate;
public String getDetails(){
return “Name:”+name+”\n”+”Salary:”+salary;
}
}
```

Invoking Overridden Methods

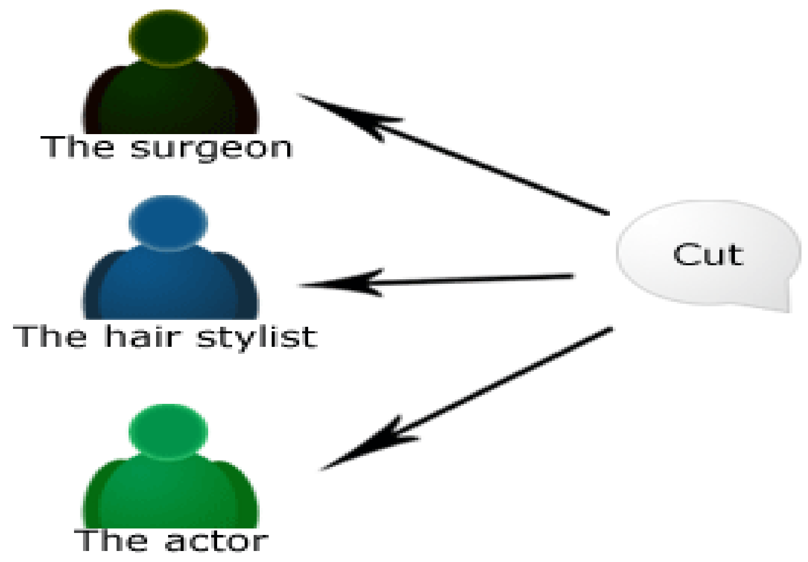
```
public class Manager extends Employee {
protected String department;
public String getDetails() {
//call parent method
return  super.getDetails()+”\nDepartment:”+ department;
}
}
```

Polymorphism

- Polymorphism is the ability to have many different forms; for example, the Manager class has access to methods from Employee class.
- An object has only one form.
- A reference variable can refer to objects of different forms.

Polymorphism

iF ANY BODY SAYS "CUT" TO THESE PEOPLE



The surgeon would begin to make an incision.

The hair stylist would begin to cut someone's hair.

The actor would abruptly stop acting out the current scene, awaiting directorial guidance.

Polymorphism

```
Employee e = new Manager(); // legal
// illegal attempt to assign Manager attribute
e.department = "Sales";
// the variable is declared as an Employee type,
// even though the Manager object has that attribute
```

Virtual Method Invocation

- Virtual method invocation is performed as follows:

  ```
  Employee e = new Manager();
  e.getDetails();
  ```
- Compile-time type and runtime type invocations have the following characteristics:
- The method name must be a member of the declared variable type; in this case Employee has a method called getDetails.
- The method implementation used is based on the runtime object's type; in this case the Manager class has an implementation of the getDetails method.

Homogeneous Collections

- Collections of objects with the same class type are called homogeneous collections. For example:

```
MyDate[] dates = new MyDate[2];

dates[0] = new MyDate(22, 12, 1964);

dates[1] = new MyDate(22, 7, 1964);
```

Heterogeneous Collections

- Collections of objects with different class types are called heterogeneous collections. For example:

```
Employee [] staff = new Employee[1024];
staff[0]= new Manager();
staff[1]= new Employee();
staff[2]= new Engineer();
```

Polymorphic Arguments

Because a Manager is an Employee, the following is valid:

```
public class TaxService {
public TaxRate findTaxRate(Employee e) {
// calculate the employee’s tax rate
}}
// Meanwhile, elsewhere in the application class
TaxService taxSvc = new TaxService();
Manager m = new Manager();
TaxRate t = taxSvc.findTaxRate(m);
```

The instanceof Operator

```
public class Employee extends Object
public class Manager extends Employee
public class Engineer extends Employee
--------------------------------------
public void doSomething(Employee e) {
if ( e instanceof Manager ) {
// Process a Manager
} else if ( e instanceof Engineer ) {
// Process an Engineer
} else {
// Process any other type of Employee
}}
```

Casting Objects

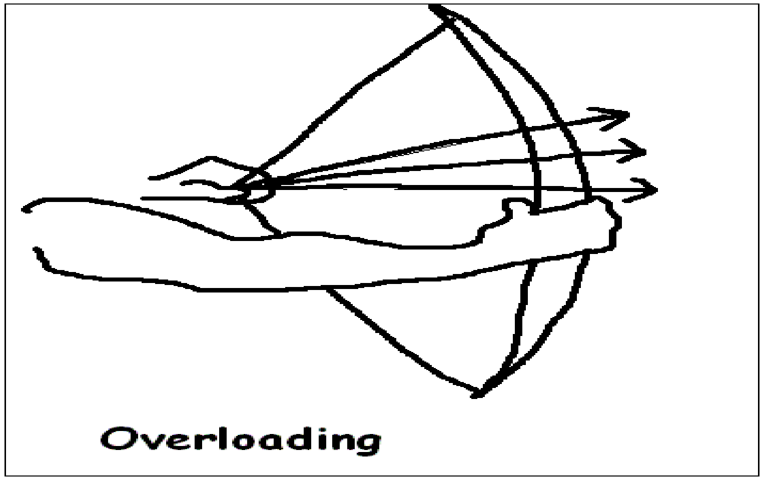
```
public void doSomething(Employee e) {
if ( e instanceof Manager ) {
Manager m = (Manager) e;
System.out.println(“This is the manager of ”
+ m.getDepartment());
}
// rest of operation
}
```

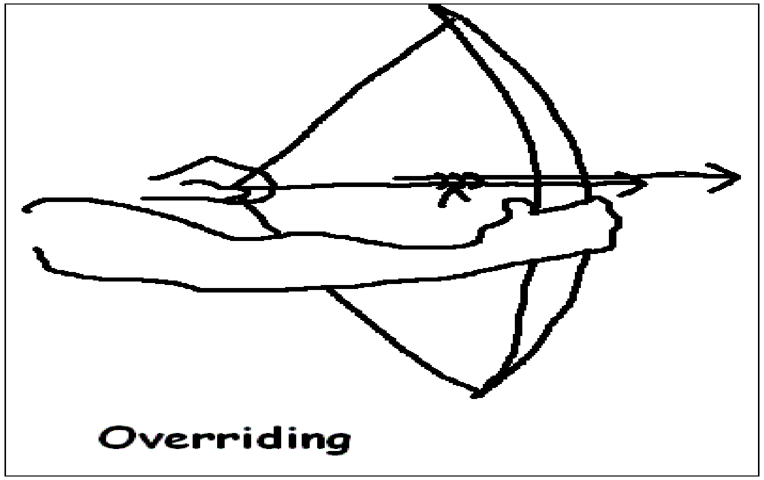
Casting Objects

- Use instanceof to test the type of an object.
- Restore full functionality of an object by casting.
- Check for proper casting using the following guidelines:
- Casts upward in the hierarchy are done implicitly.
- Downward casts must be to a subclass and checked by the compiler.
- The object type is checked at runtime when runtime errors can occur.

Overloading Methods

- Use overloading as follows:
  - public void println(int i)
  - public void println(float f)
  - public void println(String s)
- Argument lists must differ.
- Return types can be different.





Methods Using Variable Arguments

```
public float average(int... nums) {
int sum = 0;
for ( int x : nums ) {
sum += x;
}
return ((float) sum) / nums.length;
}
```

- The vararg parameter is treated as an array. For
- Example: `float gradePointAverage = stats.average(4, 3, 4);`

Overloading Constructors

- As with methods, constructors can be overloaded.

  An example is:

  public Employee(String name, double salary, Date DoB)

  public Employee(String name, double salary)

  public Employee(String name, Date DoB)
- Argument lists must differ.
- You can use the this reference at the first line of a constructor to call another constructor.

Constructors Are Not Inherited

- A subclass inherits all methods and variables from the superclass (parent class).
- A subclass does not inherit the constructor from the superclass.
  - Two ways to include a constructor are:
  - Use the default constructor.
  - Write one or more explicit constructors.

Invoking Parent Class Constructors

- To invoke a parent constructor, you must place a call to super in the first line of the constructor.
- You can call a specific parent constructor by the arguments that you use in the call to super.
- If no this or super call is used in a constructor, then the compiler adds an implicit call to super() that calls the parent no argument constructor (which could be the default constructor).

  If the parent class defines constructors, but does not provide a no-argument constructor, then a compiler error message is issued.

The Object Class

- The Object class is the root of all classes in Java.
- A class declaration with no extends clause implies extends Object. For example:

```
public class Employee {
...
}
is equivalent to:
public class Employee extends Object {
...
}
```

The Object Class

- Two important methods are:
  - equals
  - toString

The equals Method

- The == operator determines if two references are identical to each other (that is, refer to the same object).
- The equals method determines if objects are equal but not necessarily identical.
- The Object implementation of the equals method uses the == operator.
- User classes can override the equals method to implement a domain-specific test for equality.
- Note: You should override the hashCode method if you override the equals method.

An equals Example-1

```
public class MyDate {
private int day;
private int month;
private int year;
public MyDate(int day, int month, int year) {
this.day= day;
this.month = month;
this.year = year;
}
```

An equals Example-1

```
public boolean equals(Object o) {
boolean result = false;
if ( (o != null) && (o instanceof MyDate) ) {
MyDate d = (MyDate) o;
if ( (day == d.day) && (month == d.month)
&& (year == d.year) ) {
result = true;
}}
return result;
}
public int hashCode() {
return (day ^ month ^ year);
}}
```

An equals Example-2

```
class TestEquals {
public static void main(String[] args) {
MyDate date1 = new MyDate(14, 3, 1976);
MyDate date2 = new MyDate(14, 3, 1976);
if ( date1 == date2 ) {
System.out.println("date1 is identical to date2");
} else {
System.out.println("date1 is not identical to date2");
}
if ( date1.equals(date2) ) {
System.out.println("date1 is equal to date2");
} else {
System.out.println("date1 is not equal to date2");
}
```

An equals Example-2

```
System.out.println("set date2 = date1;");
date2 = date1;
if ( date1 == date2 ) {
System.out.println("date1 is identical to date2");
} else {
System.out.println("date1 is not identical to date2");
}}}
```

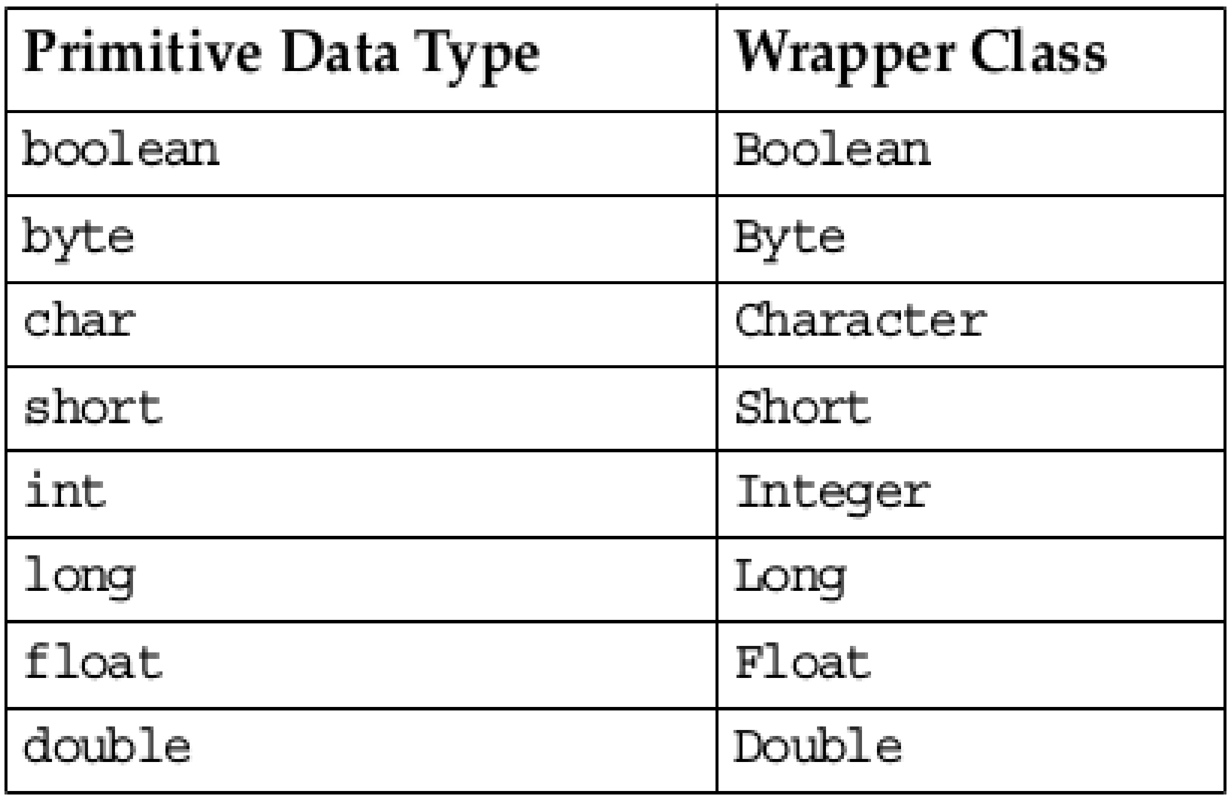
The toString Method

The toString method has the following characteristics:

- This method converts an object to a String.
- Use this method during string concatenation.
- Override this method to provide information about a user-defined object in readable format.
- Use the wrapper class’s toString static method to convert primitive types to a String.

Wrapper Classes

Look at primitive data elements as objects.



Primitive Data Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

Wrapper Classes

- An example of a wrapper class is:

```
int pInt = 420;
Integer wInt = new Integer(pInt); // this is called boxing
int p2 = wInt.intValue(); // this is called unboxing
```

- Other methods are:

```
int x = Integer.valueOf(str).intValue();
int x = Integer.parseInt(str);
```

Autoboxing of Primitive Types

Autoboxing has the following description:

- Conversion of primitive types to the object equivalent
- Wrapper classes not always needed
- Example:

```
int pInt = 420;
Integer wInt = pInt; // this is called autoboxing
int p2 = wInt; // this is called autounboxing
```

- Language feature used most often when dealing with collections
- Wrapped primitives also usable in arithmetic expressions
- Performance loss when using autoboxing

WSA | Forward looking IT finishing school

Thank you

Web Stack Academy (P) Ltd

#83, Farah Towers,

1st floor,MG Road,

Bangalore – 560001

M: +91-80-4128 9576

T: +91-98862 69112

E: info@www.webstackacademy.com

www.webstackacademy.com