OBJECT ORIENTED PROGRAMMING (OOP)

Classes and Objects

Classes and Objects

RECAP OF THE PREVIOUS SESSION

What are variables and types?

Variable?

- A variable is a data value held in memory
- Value can change during runtime
- JVM links name of variable with memory location where value is stored

Type?

- A type describes the data in basic terms
- E.g. Whole number (int) or floating-point number (double)
- The type informs JVM how much memory to set aside for a variable

Primitive Types

- A primitive type is predefined by Java language
- Java has eight primitive data types:
  - byte, short, int and long
  - float and double
  - boolean
  - char
- Check this source for more information

Remember

Java is case sensitive: be warned!

- Java keywords are lowercase (e.g. if, for, import, int)
- Class Names start with Uppercase letter
- Java uses Camel Case naming convention (check here for more information)

What is String

- In Java String is a non-primitive type, actually a list of char values
  - String can be used as a primitive type, e.g. `String name=“Antesar”;`
- Note that String starts with an uppercase letter, which in java is reserved only for class names
  - I.e. String is the name of a Class, so a String variable can be created in an object way

    `String myName = new String(“Antesar”);`
  - Where `String` is the Class name
  - `String myName` creates an object variable
  - `String()` is the class constructor method, used to instantiate the object variable
- String variables are all instances of the String class, i.e. object variables
  - This allows String methods to be directly applied to String variables, regardless which style was used to create them, e.g. `name.toUpperCase()`

Declaring an object variable: Scanner Example

- To create an scanner object: `Scanner sc = new Scanner(System.in);`
- However the above statement could be split into two parts

```
Scanner sc;
sc = new Scanner(System.in);
```

- Declaration: `Scanner sc;`
  - Creates a reference - simple variable which points to reserved memory on the heap
- Instantiation: `= new Scanner(System.in);`
  - Populates the reserved memory with values for the properties of the object
  - Transforms the reference into an object variable
  - Which can make use of the built-in coded behaviours to interact with the properties

CLASSES AND OBJECTS

Real life Objects

- In a lecture room, we can see a variety of real-life objects
  - Seating, lights, phones, students, etc.
- Some of these objects are identical in terms of properties and behaviour
  - For example, some lights have same light, colour, brightness and simply switch on and off
- A lot of objects share properties and behaviours
  - For example, two phones of same make and model, will have shared properties and behaviours
  - But they may have addition uniqueness based on stored data and installed apps
- More complex real-life object modelled, then the more unique will be its properties and behaviours
  - For example, each student will consider themselves to be unique

Classes and Objects

- To better solve real world problems through a computer program
  - We model real world objects in the program, i.e., Object Oriented Programming (OOP)
  - Applying OOP concepts of **Abstraction** and **Encapsulation**

Abstraction	Encapsulation
• Model a real-world object though the **minimum** of properties and behaviours to solve the problem • Properties specify what data can be stored • Behaviours define what actions can be done with the properties	• Objects are individual instances of a class, like String and Scanner variables • A class is the programming blueprint which defines what properties and behaviours each object instance will have • Each object will keep its properties private but provide indirect access through public behaviours

Attributes and Behaviours

Attributes

- ‘Properties’ are called attributes, or data members, instance variables or fields
- Attributes can primitive variables, non-primitive variables or object variables
- Name of attribute will depend on purpose and type
- Attributes are declared **private**
- A **static** attribute is shared by every object instance

Behaviours

- ‘Behaviours’ in Java are coded using **public** methods
  - To Initialise the attributes
  - To return value of object state, which usually start with the word ‘get’
  - To change object state
- To initialise attributes, we write one or more Constructor methods:
  - This method is used by application code to create an object instance of the class
  - Constructor method has **exactly** the same name as Class, is public and has no return type

Steps get a working program

- Create an abstract design of the (data) class
  - Focussing on minimal properties and behaviours
- Translate Abstract Design into a Code Design using Class Diagram and Pseudocode
  - Properties are replaced by variables
  - Behaviours replaced by methods
  - Apply Java naming convention and house style
- Write code for the data class and compile to ensure no syntax errors
- Write a driver application that will use the data class by:
  - Coding a main class in which objects of the data class are created
  - Invoking methods of the data class to change the state or examine the state of the objects

FAN EXAMPLE

Fan – Abstract Design

- Aim is to model a fan
  - Start by simplifying real world fan object to minimum of properties and behaviours

Properties	Behaviours	Fan example
• Minimal properties in class • Just enough to describe basic state of object • Or that required by problem	• Enough to provide basic functionality of object • And to return state of object	• Basic state is whether Fan is on or off • Two property • Behaviour is to switch Fan on or off and to set speed

Coding Design

Class Diagram

```
Fan
------------------------------------------
- isOn: Boolean
- speed: int
------------------------------------------
+ Fan()
+ turnOn(): void
+ turnOff(): void
+ setSpeed(newSpeed: int): void
+ getFanIsOn(): boolean
```

Pseudocode

- **Fan**
  1. No parameters
  2. Set field to false
- **turnOn**
  1. No parameters
  2. Set isOn vale to true
- **turnOf**
  1. No parameters
  2. Set isOn vale to false
- **getFanIsOn**
  1. No parameters
  2. Return value of the field
- **setSpeed**
  1. Integer parameter
  2. Set speed to a value

Fan.java

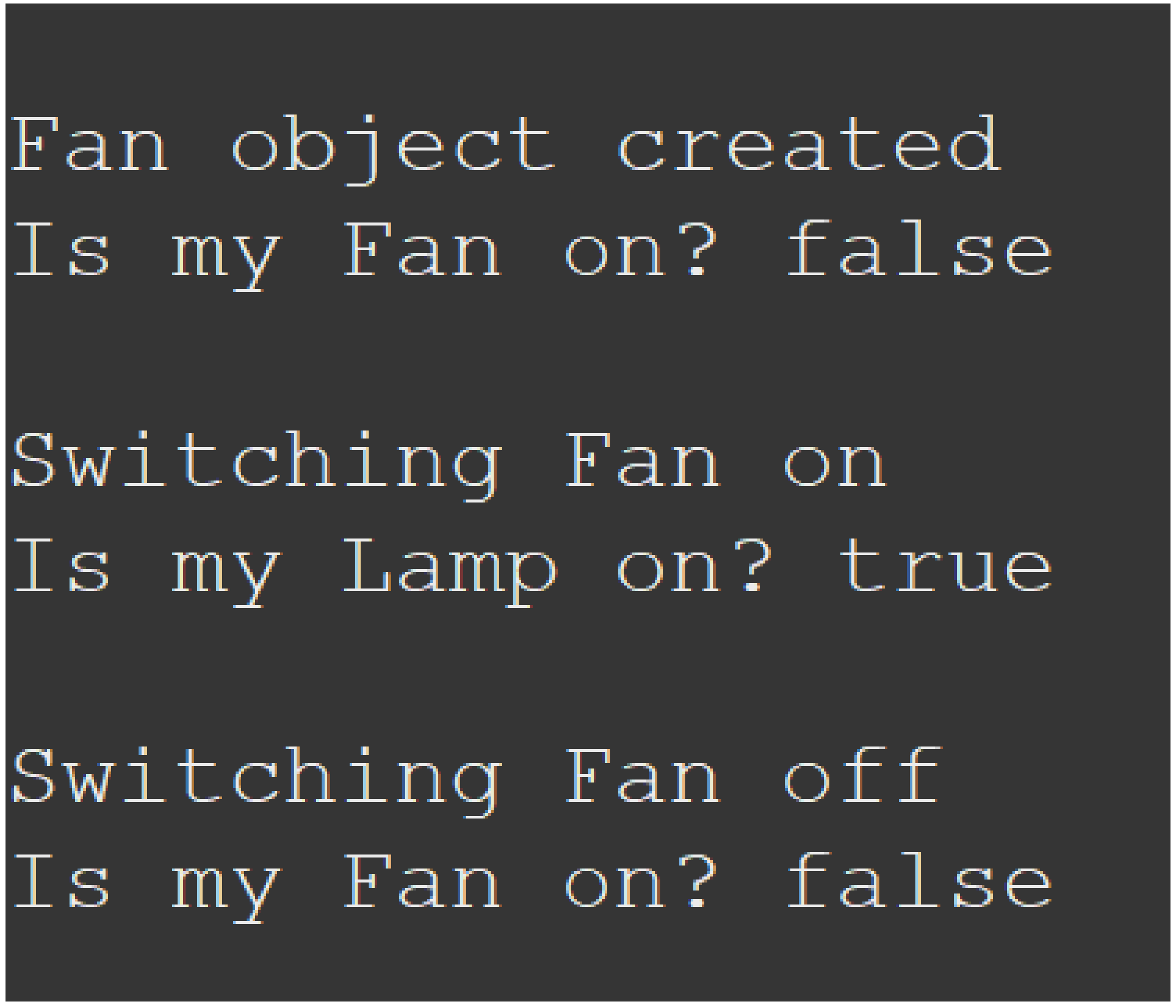
```
public class Fan{
   private boolean isOn;  // Fan's on/off state
   private int speed;  // Fan's speed (1 to 5)
   public Fan(){
      isOn = false;  // Fan is off by default
      speed = 0;  // Speed is 0 when the fan is off
   }
   public void turnOn(){
      isOn = true;
      speed = 1;  // Speed is 1 when the fan is on
(default state)
   }
public void turnOff(){
      isOn = false;
      speed = 0;
   }
   public void setSpeed(int newSpeed) {
      speed = newSpeed;
   }
   public boolean getFanIsOn(){
      return isOn;
   }
}
```

Compile the code for bugs (what do you need to test your code?)

TestFan.java

```
public class TestFan{
   public static void main(String[] args){
      //create a fan object
      Fan myFan = new Fan();

      //confirm fan object created
      System.out.println("\nFan object
      created" );

      //find out if it is on or off
      System.out.println("Is my Fan on? "
      + myFan.getFanIsOn() );

      //switch on fan
      System.out.println("\nSwitching Fan
      on");
      myFan.turnOn();
      System.out.println("Is my Fan on? "
      + myFan.getFanIsOn() );

      //switch off fan
      System.out.println("\nSwitching Fan
      off");
      myFan.turnOff();
      System.out.println("Is my Fan on? "
      + myFan.getFanIsOn() );

   } //end of the main method
} //end of the class
```

Testing Data Class



```
Fan object created
Is my Fan on? false

Switching Fan on
Is my Lamp on? true

Switching Fan off
Is my Fan on? false
```

Compile and run TestFan class to start application

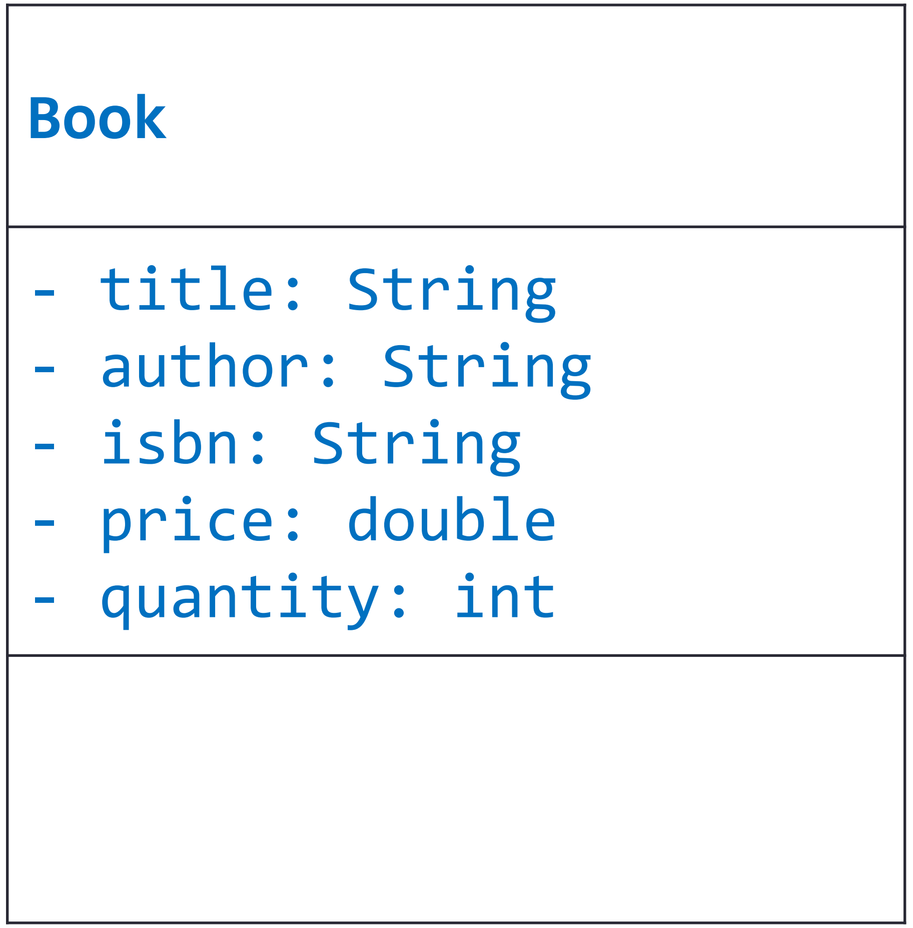
Check if Fan is on and change speed

(10 minutes)

BOOK EXAMPLE

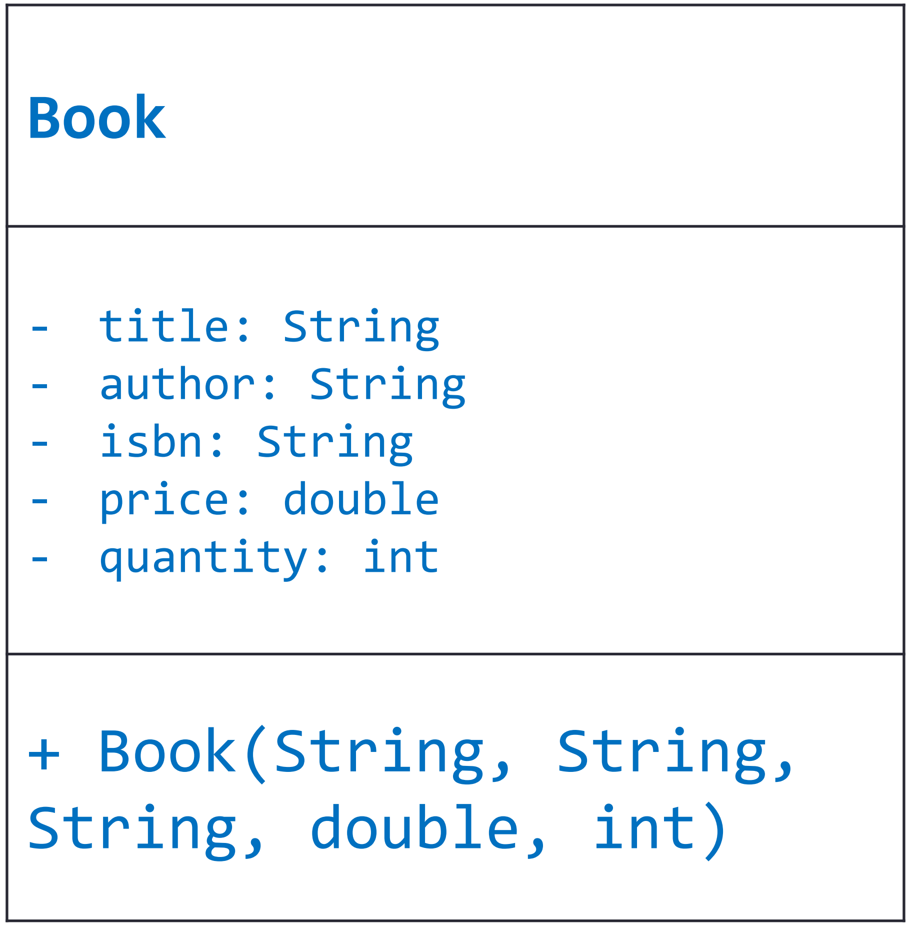
Book Example

- For a Book, there are numerous real-life properties:
  - Title, Author, Publisher, Location, Edition, Type, Pages, Price, ISBN, etc.
- Consider the Author property:
  - Should this be a String
  - Or an Array, so to store co authors
- In terms of Abstraction, we focus on the minimum of properties to achieve task:
  - Title, Author, ISBN and Price
  - But regardless if book is in a library or bookshop, we will also need to store how many we have, i.e. quantity



Constructor

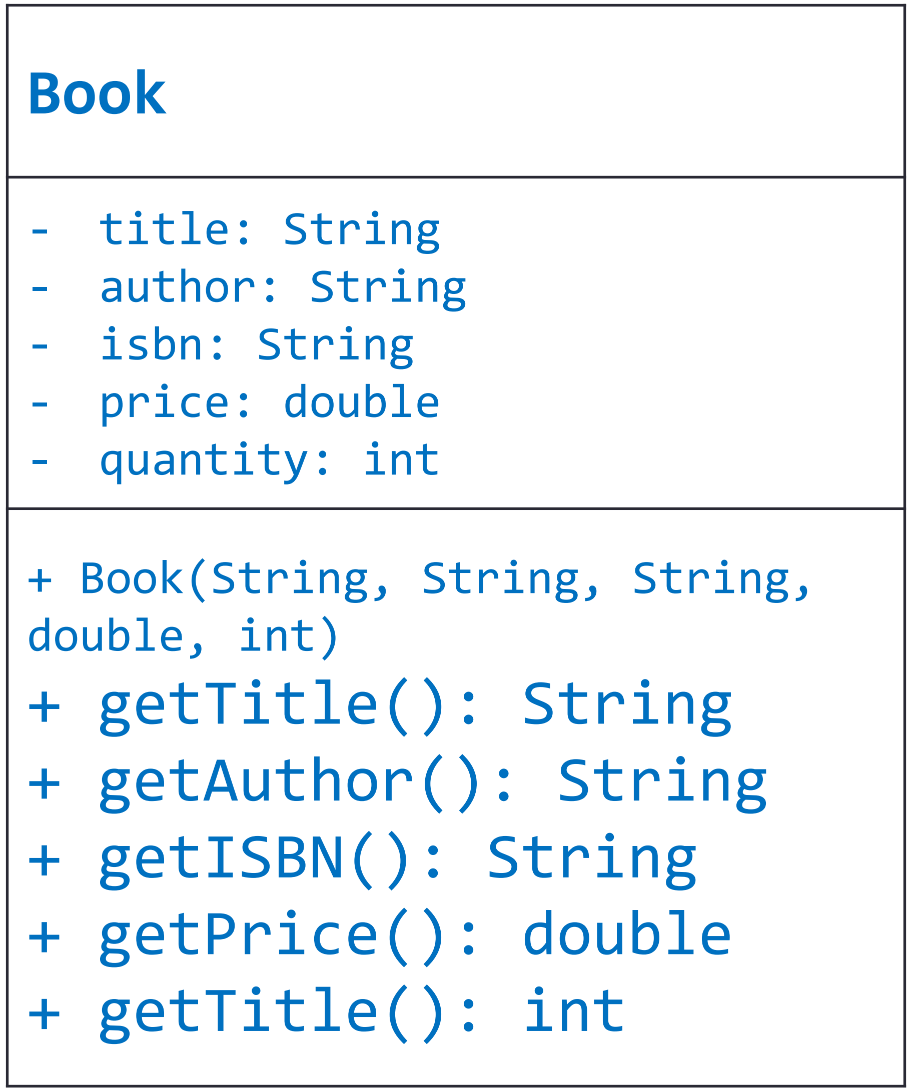
- Purpose of constructor is to set starting values of the fields
- Sometimes not all fields will need to be set
- But for Book all fields will be set



```
Book
-----------------------------
- title: String
- author: String
- isbn: String
- price: double
- quantity: int
-----------------------------
+ Book(String, String,
String, double, int)
```

Read Accessors

- Definitely need to provide read accessors for each field
  - Remember the convention is for read accessor methods to begin with the word get



```
Book

-   title: String
-   author: String
-   isbn: String
-   price: double
-   quantity: int

+ Book(String, String, String,
double, int)
+ getTitle(): String
+ getAuthor(): String
+ getISBN(): String
+ getPrice(): double
+ getTitle(): int
```

Write Accessors

- For an existing Book object
  - Only fields to change are price and quantity
  - Remember the convention is for read accessor methods to begin with the word set
- For both fields, the write accessor
  - Will receive a value to set for the field
  - Thus a parameter is required for both methods

```
Book
-------------------------------------------
-   title: String
-   author: String
-   isbn: String
-   price: double
-   quantity: int
-------------------------------------------
+ Book(String, String, String,
double, int)
+ getTitle(): String
+ getAuthor(): String
+ getISBN(): String
+ getPrice(): double
+ getTitle(): int
+ setPrice(double):void
+ setQuantity(int):void
```

Book Class Diagram

- In this example, there are no service or helper methods
  - For example, no method to return price as a string (fixed to two decimal places)
- The question of validation always arises when including set methods
  - In OOP, validation will be matter for the Test/Main class

```
Book
---------------------------------------------------------
-  title: String
-  author: String
-  isbn: String
-  price: double
-  quantity: int
---------------------------------------------------------
+ Book(String, String, String, double, int)
+ getTitle(): String
+ getAuthor(): String
+ getISBN(): String
+ getPrice(): double
+ getTitle(): int
+ setPrice(double):void
+ setQuantity(int):void
```

Book Class Code

```
public class Book{
   //fields
   private String title;
   private String author;
   private String isbn;
   private double price;
   private int quantity;

   //constructor
   public Book(String title,
   String author, String isbn, double price,
   int quantity){
      this.title = title;
      this.author = author;
      this.isbn = isbn;
      this.price = price;
      this.quantity = quantity;
   }

   //read accessors
   public String getTitle(){return title;}
   public String getAuthor(){return author;}
   public String getISBN(){return isbn;}
   public double getPrice(){return price;}
   public int getQuantity(){return
   quantity;}

   //set accessors
   public void setPrice(double price){
      this.price=price;
   }

   public void getQuantity(int quantity){
      this.quantity=quantity;
   }
}//end of class
```

This keyword

- Scope is an issue with classes within a class
  - Fields are declared at class level so are accessible inside the methods
- Methods with parameters such as the constructor are likely to name parameters after the fields
  - For example in the constructor, we have the field named title and the parameter named title
- If we use the identifier title within constructor method Java will assume this refers to the parameter
  - If we want Java to use the field, we use the this keyword, e.g., `this.title`
- Given the statement `this.title = title;` in the constructor
  - `this.title` refers to the class level field
  - title refers to the local parameter

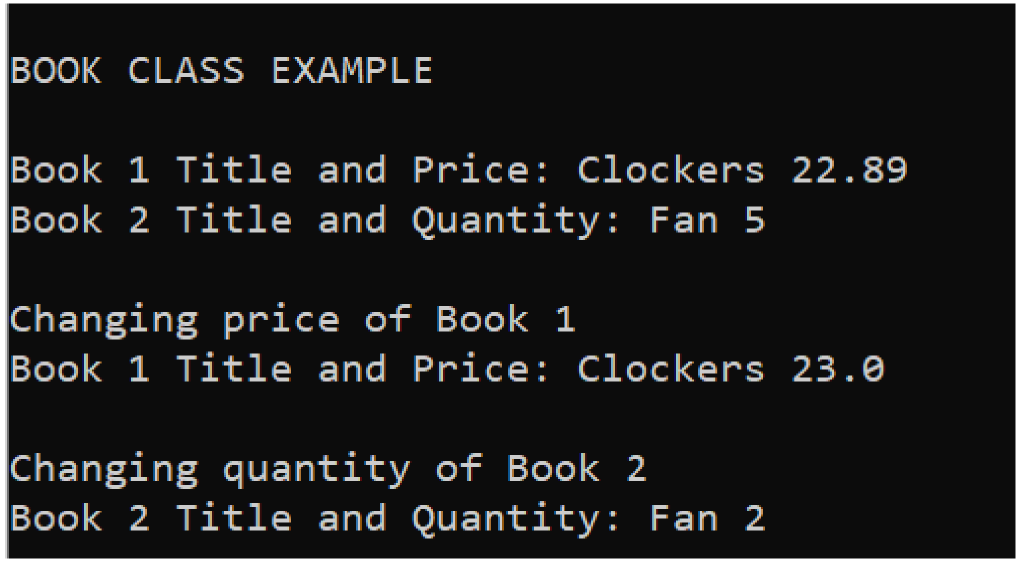
TestBook.java (1)

```
public class TestBook{
   public static void main(String[] args){
      //create a Book`objects
      Book book1 = new Book("Clockers", "R Price","978-0395537619",22.89,10);
      Book book2 = new Book("Fan", "D Rhodes", "978-1909807808", 9.92, 5);

      //Display Program Title
      System.out.println("\nBOOK CLASS EXAMPLE" );

      //display state
      System.out.println("\nBook 1 Title and Price: " + book1.getTitle() + " " +
      book1.getPrice());
      System.out.println("Book 2 Title and Quantity: " + book2.getTitle() + " " +
      book2.getQuantity());
```

TestBook.java (2)

```
    //change price for book 1
    book1.setPrice(23);
    System.out.println("\nChanging price of Book 1");
    System.out.println("Book 1 Title and Price: " + book1.getTitle() + " " +
    book1.getPrice());

    //change quantity for book 2
    book2.setQuantity(2);
    System.out.println("\nChanging quantity of Book 2");
    System.out.println("Book 2 Title and Quantity: " + book2.getTitle() + " " +
    book2.getQuantity());

  } //end of the main method
} //end of the class
```

Testing the Book Class



```
BOOK CLASS EXAMPLE

Book 1 Title and Price: Clockers 22.89
Book 2 Title and Quantity: Fan 5

Changing price of Book 1
Book 1 Title and Price: Clockers 23.0

Changing quantity of Book 2
Book 2 Title and Quantity: Fan 2
```

- Compile and run TestBook class to start application
- Use different test values
- (10 minutes)

CLASS WRAPPERS AND SUPER CLASS IN JAVA

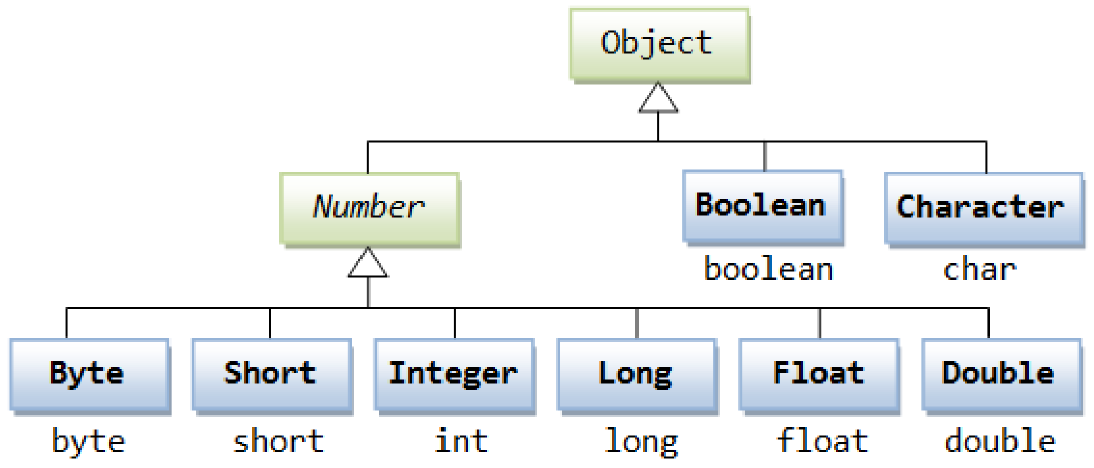
Java Wrapper Classes

- Wrapper classes provide a way to use primitive data types as objects.
- Sometimes an object variable is required by some code, construct or method
  - Meaning a primitive type variable cannot be used
  - In such circumstances the equivalent wrapper class object is used
- Example creation of a Wrapper object

```
int num = 20;
Integer objNum = new Integer(20)
Integer objNum = Integer.valueOf(num)
Integer objNum = num
```

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

Java Super Class



Note that *Number* is an abstract class

- All classes provided by Java Language:
  - Are derived directly or indirectly from the super class `Object`
- When an object variable of the Object class is declared
  - It can by instantiated by any derived class
  - E.g. `Object obj = new Integer(8)`
  - Or `Object obj = new Double(3.14)`
  - Or even `Object obj = new String(“ABC”);`

Task 1

- Write a Java program that asks the user to input a sentence. Your task is to perform the following operations on the sentence using String methods:
    1. Convert the sentence to all uppercase letters
    2. Count the number of words in the sentence
    3. Reverse the entire sentence
    4. Check if the sentence contains the word "Java" (case-insensitive)
    5. Replace all spaces in the sentence with underscores ("_")
- Task:
- Prompt the user to input a sentence
- Perform the above operations in the order listed
- Display the original sentence, the modified sentence after each operation, and the final result

- Hints (Methods to Use):
    - To read a sentence:scanner.nextLine();
    - To convert to uppercase: toUpperCase()
    - To split the sentence into words: split(" ") (for spaces)
    - To reverse the sentence, you can convert it to a character array or use StringBuilder with reverse()
    - To check if a sentence contains a word: contains() and toLowerCase() (for case-insensitive search)
    - To replace spaces with underscores: replace(" ", "_")

Task 2: Employee Object Design

- Model Employee object
- Design code and write pseudocode
- Write Java code and test it using your preferred IDE