Trace Code

```java
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;
```

Declare myCircle

myCircle no value
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;
Trace Code, cont.

```java
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;
```

Assign object reference to myCircle

```
myCircle
: Circle
radius: 5.0
```

reference value
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;
Trace Code, cont.

Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

Assign object reference to yourCircle

myCircle reference value

: Circle
radius: 5.0

yourCircle reference value

: Circle
radius: 1.0
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

Change radius in yourCircle
Classes and Objects

- A Java program consists of one or more classes.
- A class is an abstract description of objects.
- Here is an example class:
  ```java
  class Dog { ...description of a dog goes here... }
  ```
- Here are some objects of that class:

Ravi Kant Sahu, Asst. Professor © Lovely Professional University, Punjab (India)

Fig. 1: Instantiating two Trees from the Tree class
Purpose of Variables

• A variable contains information
• Variables are used to provide information to the program so that it can perform its tasks
Providing access to an object’s information

Two different ways to provide information through variables:

• Direct access

• Indirect access
Direct Access

The variables have public access and they are directly accessible (= exposed) to the user.

Example:

```java
public class BankAccount {
    ....
    public double balance;
    ....
}
```

The variables balance can be accessed directly from anywhere (due to the public access specifier):

```java
public class MyProgram {
    public static void main( String[] args ){
        BankAccount stu1 = new BankAccount(123, "John", 1000.0);
        stu1.balance = stu1.balance + 500;
    }
}
```
Indirect Access

The variables have private access and they indirectly accessible (= hidden) to the user through public methods

Example:

```java
public class BankAccount {
    private double balance;
    ....
    public double getBalance()
    {
        return balance;
    }
    public void putBalance(double x)
    {
        balance = x;
    }
}
```

Balance must be accessed indirectly through methods getBalance() and putBalance():

```java
public class MyProgram {
    public static void main( String[] args ) {
        BankAccount stu1 = new BankAccount(123, "John", 1000.0);
        double x;
        x = stu1.getBalance();
        stu1.putBalance(x + 500);
    }
}
```
Decoupling Implementation and Functionality

There are 2 aspects in the process of providing information

• Implementation = How is the data stored that are used to provide the information.
• Functionality = What operations (= functionality) are allowed on the information.

Effect of the direct data access on these 2 aspects:
When information is realized using the direct access technique, we say that the functionality and implementation are coupled together.

Effect of the indirect data access on these 2 aspects:
When information is realized using the indirect access technique, we say that the functionality and implementation are decoupled.
Terminology

**Information hiding** = "hide" the variables (= information) from the users so they can not accesses the variable directly

- This is done by having **private** access variables
- The user must make use of **public** access methods to gain access to the variables

**Abstract Data Type (ADT)** = a data storage structure that is defined by the functionality

- The **functionality** is defined by *what kind of information* can you obtain from the ADT.
- The ADT will provide a number of **public** access methods to provide the functionalities
- To **ensure data integrity** (the correct information is provide), the variables used to store the information are protected by **private** access permission
Take-home point

Information hiding and Abstract Data Type are in fact the same technique.

Functionality and implementation are decoupled when using Information Hiding and Abstract Data Type.

They (= Information Hiding and ADT) both use:

• private access instance variables to hide the data from the users

• Users must make use of public access methods to gain access to the information.
Advantage of ADT

ADT can separate

• implementation of the information (= how the information is stored)
• functionality of the information (= what information will be provided)
Providing age information
Implementation 1: use a variable `int age` to store the age of a person

**Direct access solution:**

```java
public class AgeInformation {
    public String name;
    public int age;

    // Constructor
    AgeInformation(String s, int x) {
        name = s;
        age = x;
    }

    // Constructor
    AgeInformation(String s, int x) {
        name = s;
        age = x;
    }

    public int getAge() {
        return age;
    }

    // Constructor
    AgeInformation(String s, int x) {
        name = s;
        age = x;
    }

    // Constructor
    AgeInformation(String s, int x) {
        name = s;
        age = x;
    }

    public int getAge() {
        return age;
    }
}
```

**ADT solution:**

```java
public class AgeInformation {
    private String name;
    private int age;

    // Constructor
    AgeInformation(String s, int x) {
        name = s;
        age = x;
    }

    // Constructor
    AgeInformation(String s, int x) {
        name = s;
        age = x;
    }

    public int getAge() {
        return age;
    }

    // Constructor
    AgeInformation(String s, int x) {
        name = s;
        age = x;
    }

    // Constructor
    AgeInformation(String s, int x) {
        name = s;
        age = x;
    }

    public int getAge() {
        return age;
    }
}
```
Implementation 1 continued

Access via direct access solution

```java
public class MyProgram {
    public static void main(String[] a) {
        AgeInformation stu1 = new AgeInformation("John", 22);

        int info;
        info = stu1.age;
        ...
    }
}
```

Access via ADT solution

```java
public class MyProgram {
    public static void main(String[] a) {
        AgeInformation stu1 = new AgeInformation("John", 22);

        int info;
        info = stu1.getAge();
        ...
    }
}
```
Providing age information

Implementation 2: We use 3 variables `int day, month, year` to store the birthday of a person. The age is computed using the current system time.

**Direct access solution:**

```java
public class AgeInformation {
    public String name;
    public int day, month, year;

    // Constructor
    AgeInformation(String s, int a, int b, int c) {
        name = s;
        day = a;
        month = b;
        year = c;
    }
    ...
}
```

**ADT solution:**

```java
public class AgeInformation {
    public String name;
    public int day, month, year;

    // Constructor
    AgeInformation(String s, int a, int b, int c) {
        name = s;
        day = a;
        month = b;
        year = c;
    }

    public int getAge() {
        // Read system time, compute age
        // (Algorithm omitted)
        return age;
    }
}
```
Implementation 2 continued

Access via direct access solution

```java
public class MyProgram {
    public static void main(String[] a) {
        AgeInformation stu1 = new AgeInformation("John", 22);

        int info;
        info = stu1.age;
        // NO LONGER WORKS!!
        ...
    }
}
```

Access via ADT solution

```java
public class MyProgram {
    public static void main(String[] a) {
        AgeInformation stu1 = new AgeInformation("John", 22);

        int info;
        info = stu1.getAge();
        // STILL WORKS!!
        ...
    }
}
```