

# Method types and return values

server

```
public class Clock {
    private int hours, minutes, seconds;
    ...
    public void hourElapsed() {
        hours = (hours + 1) % 24;
    }
    ...
    public int getHours() {
        return hours;
    }
    ...
    public String toString() {
        return (hours + "\t" + minutes + "\t" + seconds);
    }
    ...
}
```

client

```
public class SomeClass {
    ...
    Clock c = new Clock(0,0,0);

    // Invoking a void method
    c.hourElapsed(2);

    // Invoking typed methods
    int h = c.getHours();
    System.out.println(c.toString());
    ...
}
```

Void method: has no type and no return value

Typed method: has either a primitive type or an object type.  
Returns a value that must conform to the method's type.

# Accessor methods

---

server

```
public class Clock {
    private int hours, minutes, seconds;

    ...

    public int getHours() {
        return hours;
    }

    public int getMinutes() {
        return minutes;
    }

    public int getSeconds() {
        return seconds;
    }

    ...
}
```

Accessor methods: used to query and return information about private variables

Enable controlled access to private variables from the outside.

client

```
public class someClass {
    ...
    Clock c = new Clock(12,0,0);
    ...
    int h = c.getHours();
    ...
    System.out.print(c.getSeconds())
    ...
}
```

# Mutator methods

server

```
public class Clock {
    private int hours, minutes, seconds;

    ...

    public void setHours(int hours) {
        if ((hours > 0) && (hours <= 24))
            this.hours = hours;
    }

    public void setMinutes(int minutes) {
        if ((minutes >= 0) && (minutes < 60))

            this.minutes = minutes;
    }

    public void setSeconds(int seconds) {
        if ((seconds >= 0) && (seconds < 60))
            this.seconds = seconds;
    }

    ...
}
```

Mutator methods: used to set the values of private variables

Enable controlled update of private variables from the outside.

client

```
public class someClass {
    ...
    int deadlineHour = 20;
    ...
    Clock c = new Clock(12,0,0);
    ...
    c.setHours(17);
    ...
    c.setHours(deadlineHour-1);
    ...
    c.setMinutes(scan.nextInt());
    ...
}
```

# Variable life cycle

## Static variables

## Instance variables

## Local variables

- Static variables: created the first time a method from the class is invoked
- Instance variables: created when the object is created and recycled when the object is destroyed
- Local variables: created when the method is invoked and recycled when it returns
- Parameters: same as local variables. Initialized by the arguments supplied by the caller.

```
public class BankAccount {  
    private static int nextAccountNumber = 1;  
    private static int bankDeposits = 0;  
  
    private int number;  
    private String owner;  
    private int balance;  
  
    // Sets up a new account  
    public BankAccount (String owner, int balance) {  
        this.number = nextAccountNumber++;  
        this.owner = owner;  
        this.balance = balance;  
    }  
  
    // Handles a withdrawal  
    public void withdraw (int amount) {  
        int balanceTemp = balance - amount  
        if (balanceTemp >= 0) {  
            balance = balanceTemp;  
            bankDeposits -= amount;  
        }  
        // else ... omitted  
    }  
}
```

## parameter

## Standard class methods

---

```
// Returns a textual object description
public String toString () {
    return (number + "\t" + owner + "\t" + (int) balance);
}

// Compares this object to another given object
public boolean equals(BankAccount other) {
    if (other == null) {
        return false;
    }
    // Two bank accounts are equal if their account number is the same
    return (this.number == other.getNumber());
}

// Returns a copy of this object
public BankAccount clone() {
    BankAccount copy = new BankAccount(this.owner, this.balance);
    copy.setNumber(this.number);
    nextAccountNumber--; // restore the "wasted" account number
    return copy;
}
}
```

# Standard class methods

```
public class BankAccountDemo {  
  
    public static void main(String[] args) {  
  
        BankAccount tomAct = new BankAccount("Tom", 500);  
        BankAccount bobAct = new BankAccount("Bob", 200);  
        BankAccount p;  
        System.out.println(tomAct.toString());  
        System.out.println(bobAct);  
        System.out.println();  
  
        tomAct.deposit(1000);  
        tomAct.transferTo(400, bobAct);  
        p = bobAct;  
  
        System.out.println(tomAct);  
        System.out.println(bobAct);  
        System.out.println(p);  
        System.out.println();  
  
        p = tomAct.clone();  
        System.out.println(p);  
        System.out.println();  
  
        BankAccount ronAct = new BankAccount("Ron", 0);  
        System.out.println(ronAct);  
        System.out.println(tomAct.equals(ronAct));  
        System.out.println(tomAct.equals(p));  
    }  
}
```

```
F:\demo>javac BankAccountDemo.java  
F:\demo>java BankAccountDemo  
1      Tom      500  
2      Bob      200  
  
1      Tom      1093  
2      Bob      598  
2      Bob      598  
  
1      Tom      1093  
  
3      Ron      0  
false  
true
```