COM 3529
Software Testing & Analysis

Professor Phil McMinn

3.1 White Box Coverage Criteria based on
Data Flow Analysis

Definition

```
int x = y;

public int method (int a) {
 // …
}
```

Definition

```
int x = y;

public int method (int a) {
 // …
}
```

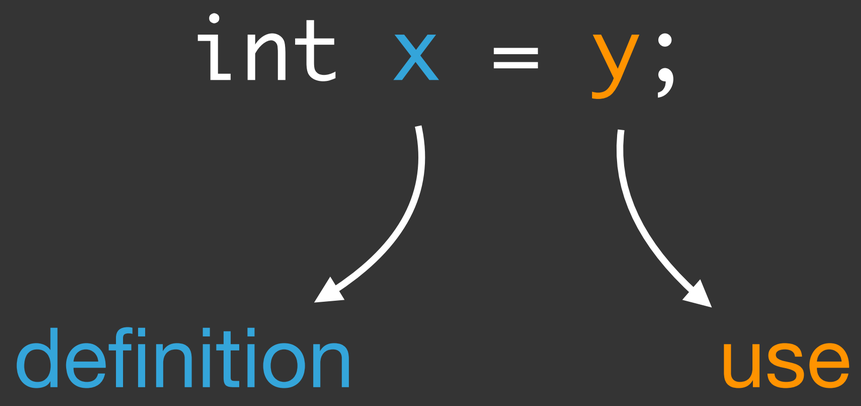
Use

```
int x = y;

System.out.println(“Hello ” + name);

o.update();

if (a > b) {
  // …
}

return result;
```

Use

```
int x = y;

System.out.println(“Hello ” + name);

o.update();

if (a > b) {
  // …
}

return result;
```

Definitions *and* Uses



Definitions *and* Uses

```
int x = y;
```

definition use

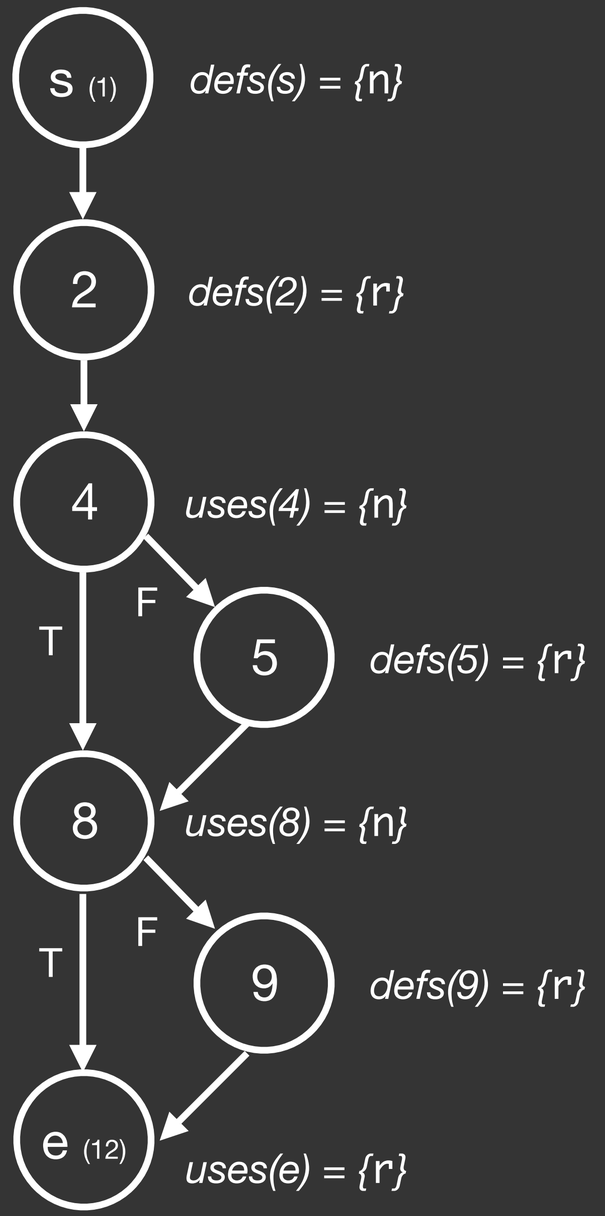
```
x = x + 1;
```

definition use

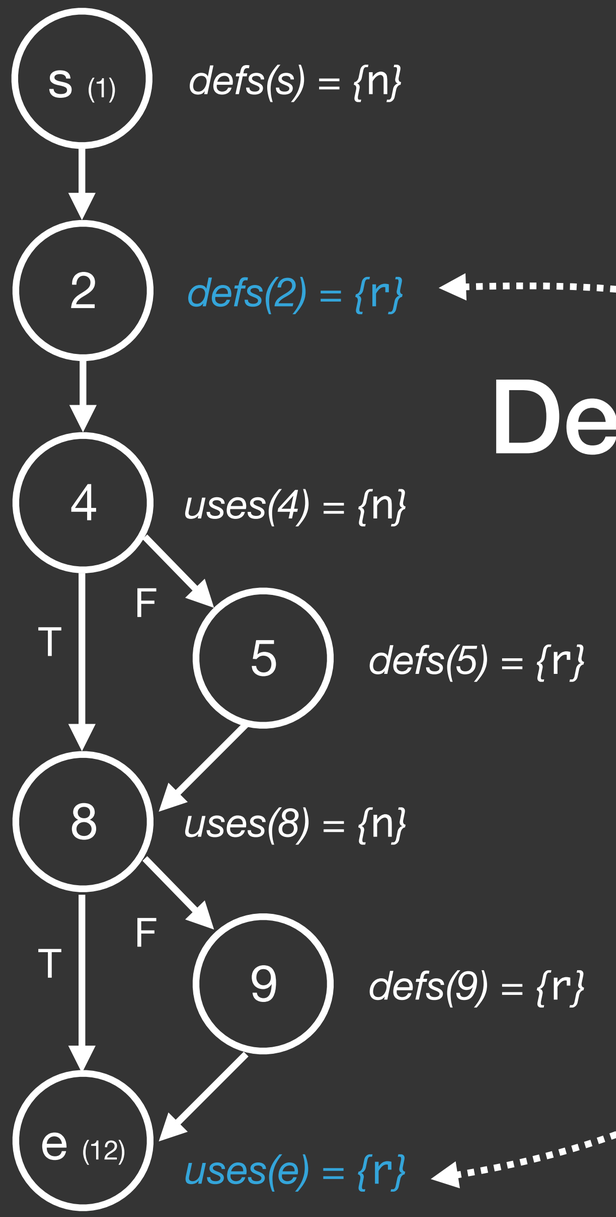
```
x += 1;
```

```
x ++;
```

```java
1     public static int sign(int n) {
2         int r = 0;
3
4         if (n > 0) {
5             r = 1;
6         }
7
8         if (n < 0) {
9             r = -1;
10        }
11
12        return r;
13    }
```

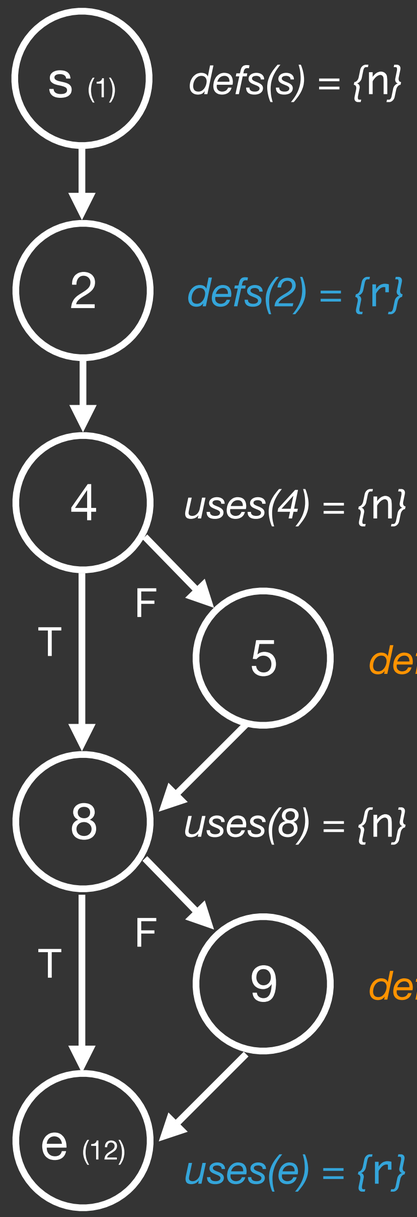


```java
1     public static int sign(int n) {
2         int r = 0;
3
4         if (n > 0) {
5             r = 1;
6         }
7
8         if (n < 0) {
9             r = -1;
10        }
11
12        return r;
13    }
```



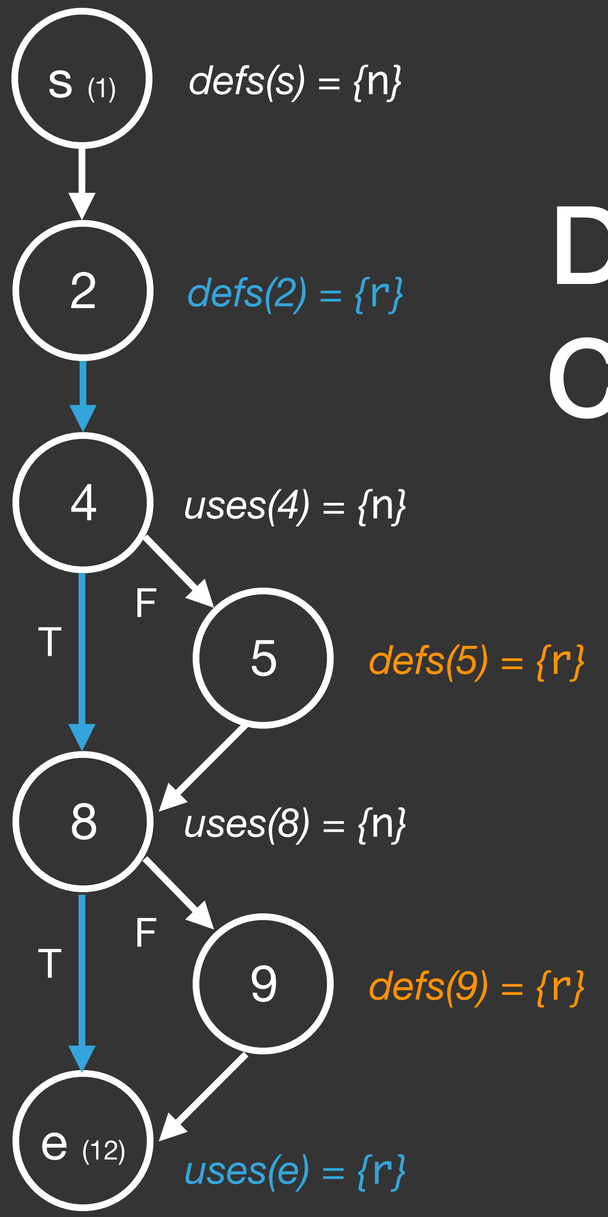
Definition-Use Pair

```
1     public static int sign(int n) {
2         int r = 0;
3
4         if (n > 0) {
5             r = 1;
6         }
7
8         if (n < 0) {
9             r = -1;
10        }
11
12        return r;
13    }
```



Killing Definitions

```
1     public static int sign(int n) {
2         int r = 0;
3
4         if (n > 0) {
5             r = 1;
6         }
7
8         if (n < 0) {
9             r = -1;
10        }
11
12        return r;
13    }
```

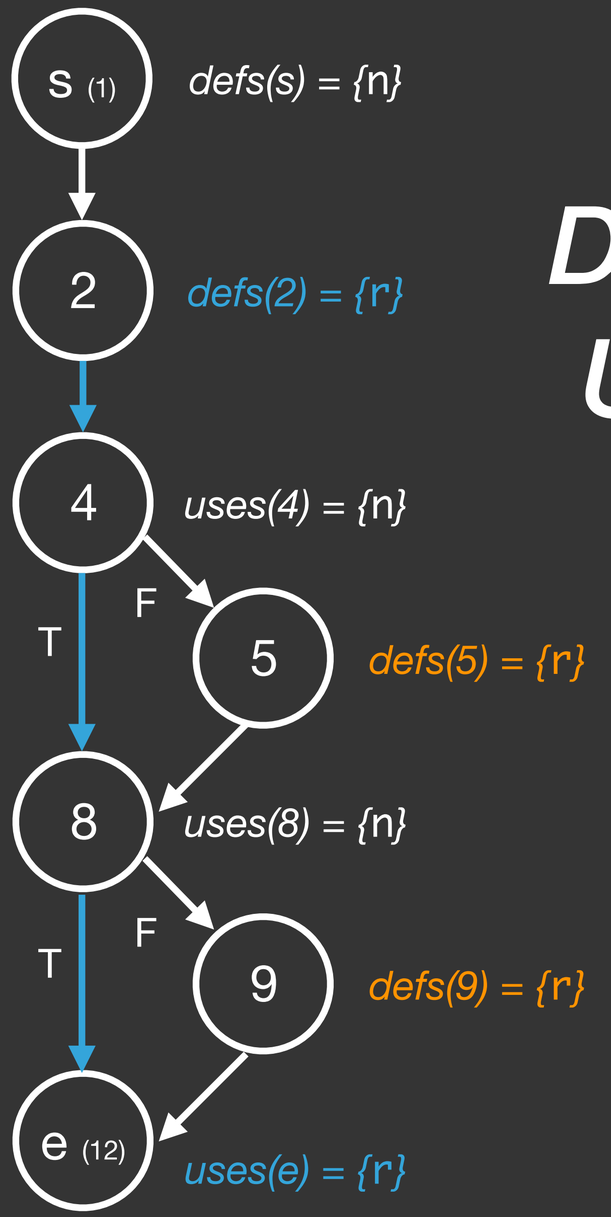


Definition-Clear Path

The definition *reaches* the use

Formally, a path from n_i to n_j is definition-clear with respect to a variable v if for each node n_k on the path between n_i and n_j, (i.e., $n_k \neq n_i \wedge n_k \neq n_j$), $v \notin \mathit{defs}(n_k)$. That is, none of the nodes between n_i and n_j is a killing definition. If a definition-clear path exists from a definition of v at n_i to a use of $\mathtt{v}$ at n_j, the definition of v is said to **reach** the use at n_j.

```
1     public static int sign(int n) {
2         int r = 0;
3
4         if (n > 0) {
5             r = 1;
6         }
7
8         if (n < 0) {
9             r = -1;
10        }
11
12        return r;
13    }
```

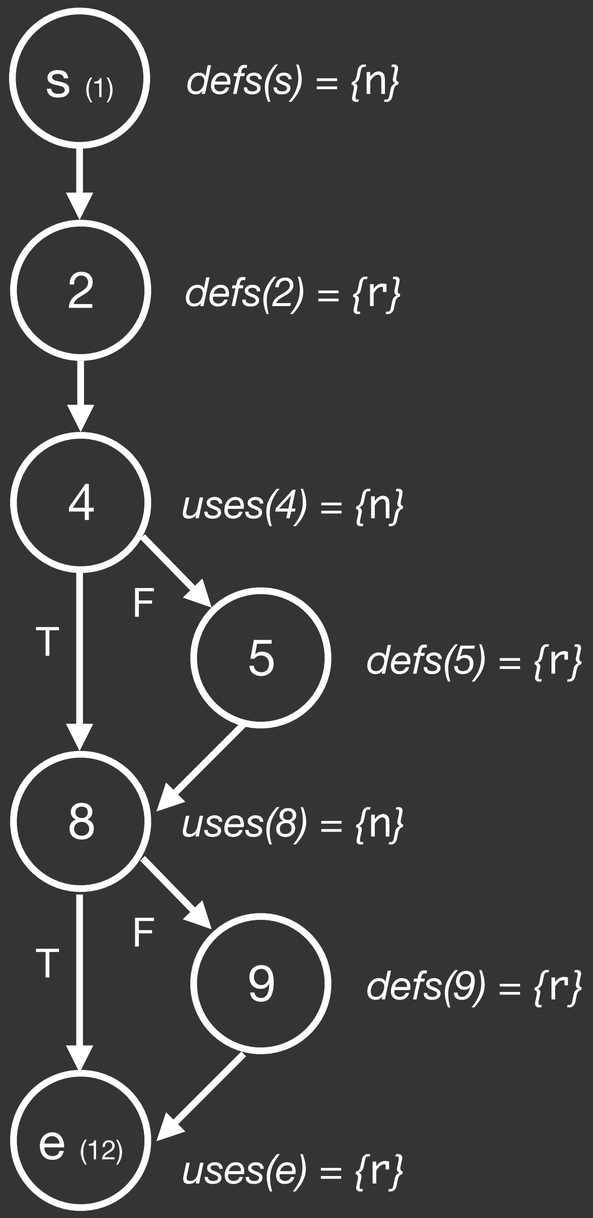


Definition-Use Path

```java
1   public static int sign(int n) {
2       int r = 0;
3
4       if (n > 0) {
5           r = 1;
6       }
7
8       if (n < 0) {
9           r = -1;
10      }
11
12      return r;
13  }
```

The set *DU*

No.	Variable	Definition	Use	Definition-Use Path
1	r	2	e	2 → 4 → 8 → e
2	r	5	e	5 → 8 → e
3	r	9	e	9 → e
4	n	s	4	s → 2 → 4
5	n	s	8	s → 2 → 4 → 8
6	n	s	8	s → 2 → 4 → 5 → 8



All Defs Coverage

Each definition reaches at least one use of the same variable

No.	Variable	Definition	Use	Definition-Use Path
1	r	2	e	$2 \rightarrow 4 \rightarrow 8 \rightarrow e$ ✓
2	r	5	e	$5 \rightarrow 8 \rightarrow e$ ✓
3	r	9	e	$9 \rightarrow e$ ✓
4	n	s	4	$s \rightarrow 2 \rightarrow 4$ ✓
5	n	s	8	$s \rightarrow 2 \rightarrow 4 \rightarrow 8$
6	n	s	8	$s \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 8$

All Uses Coverage

Each definition reaches each use of the same variable

No.	Variable	Definition	Use	Definition-Use Path
1	`r`	2	e	$2 \rightarrow 4 \rightarrow 8 \rightarrow e$ ✓
2	`r`	5	e	$5 \rightarrow 8 \rightarrow e$ ✓
3	`r`	9	e	$9 \rightarrow e$ ✓
4	`n`	s	4	$s \rightarrow 2 \rightarrow 4$ ✓
5	`n`	s	8	$s \rightarrow 2 \rightarrow 4 \rightarrow 8$ ✓
6	`n`	s	8	$s \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 8$

All Def-Use Path Coverage

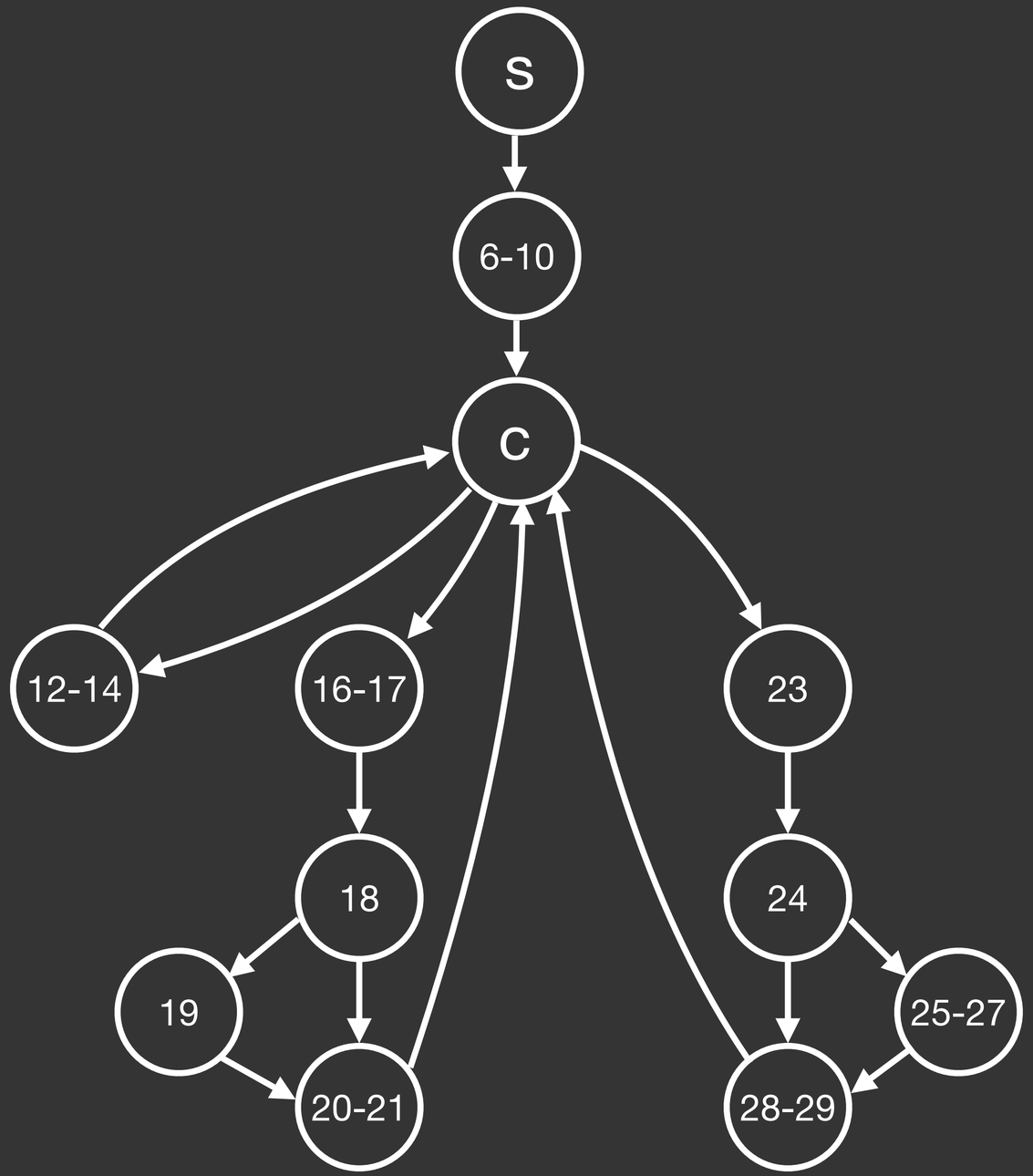
Every path in *DU* needs to be executed

No.	Variable	Definition	Use	Definition-Use Path
1	`r`	2	e	$2 \rightarrow 4 \rightarrow 8 \rightarrow e$ ✓
2	`r`	5	e	$5 \rightarrow 8 \rightarrow e$ ✓
3	`r`	9	e	$9 \rightarrow e$ ✓
4	`n`	s	4	$s \rightarrow 2 \rightarrow 4$ ✓
5	`n`	s	8	$s \rightarrow 2 \rightarrow 4 \rightarrow 8$ ✓
6	`n`	s	8	$s \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 8$ ✓

When Should You Use Data-Flow Testing?

```java
public class VendingMachine {

    private int totalCoins, currentCoins;
    private boolean allowVend;

    public VendingMachine() {
        totalCoins = 0;
        currentCoins = 0;
        allowVend = false;
    }

    public void returnCoins() {
        currentCoins = 0;
    }

    public void addCoin() {
        currentCoins ++;
        if (currentCoins > 1) {
            allowVend = true;
        }
    }

    public void vend() {
        if (allowVend) {
            totalCoins += currentCoins;
            currentCoins = 0;
            allowVend = false;
        }
    }
}
```

```java
public class VendingMachine {

    private int totalCoins, currentCoins;
    private boolean allowVend;

    public VendingMachine() {
        totalCoins = 0;
        currentCoins = 0;
        allowVend = false;
    }

    public void returnCoins() {
        currentCoins = 0;
    }

    public void addCoin() {
        currentCoins ++;
        if (currentCoins > 1) {
            allowVend = true;
        }
    }

    public void vend() {
        if (allowVend) {
            totalCoins += currentCoins;
            currentCoins = 0;
            allowVend = false;
        }
    }
}
```



```java
public class VendingMachine {

    private int totalCoins, currentCoins;
    private boolean allowVend;

    public VendingMachine() {
        totalCoins = 0;
        currentCoins = 0;
        allowVend = false;
    }

    public void returnCoins() {
        currentCoins = 0;
    }

    public void addCoin() {
        currentCoins ++;
        if (currentCoins > 1) {
            allowVend = true;
        }
    }

    public void vend() {
        if (allowVend) {
            totalCoins += currentCoins;
            currentCoins = 0;
            allowVend = false;
        }
    }
}
```