

Lecture 10 - June 5

TDD with JUnit

JUnit Test Method vs. Method Under Test
TDD & Regression Testing
JUnit Test: An Exception Expected or Not

Announcements/Reminders

- Today's class: notes template posted
- **ProgTest1** next Friday JUN 13
- **ProgTest1 guide** (policies & requirements) released
 - + **PracticeTest1** & **Survey** on Review Session Time
 - + **ProgTest0** marks & feedback (or MON, JUN 9 latest)
- Priorities:
 - + **Lab1** solution, **Lab2**
 - + Slides on Classes and Objects
 - + Slides on Exceptions

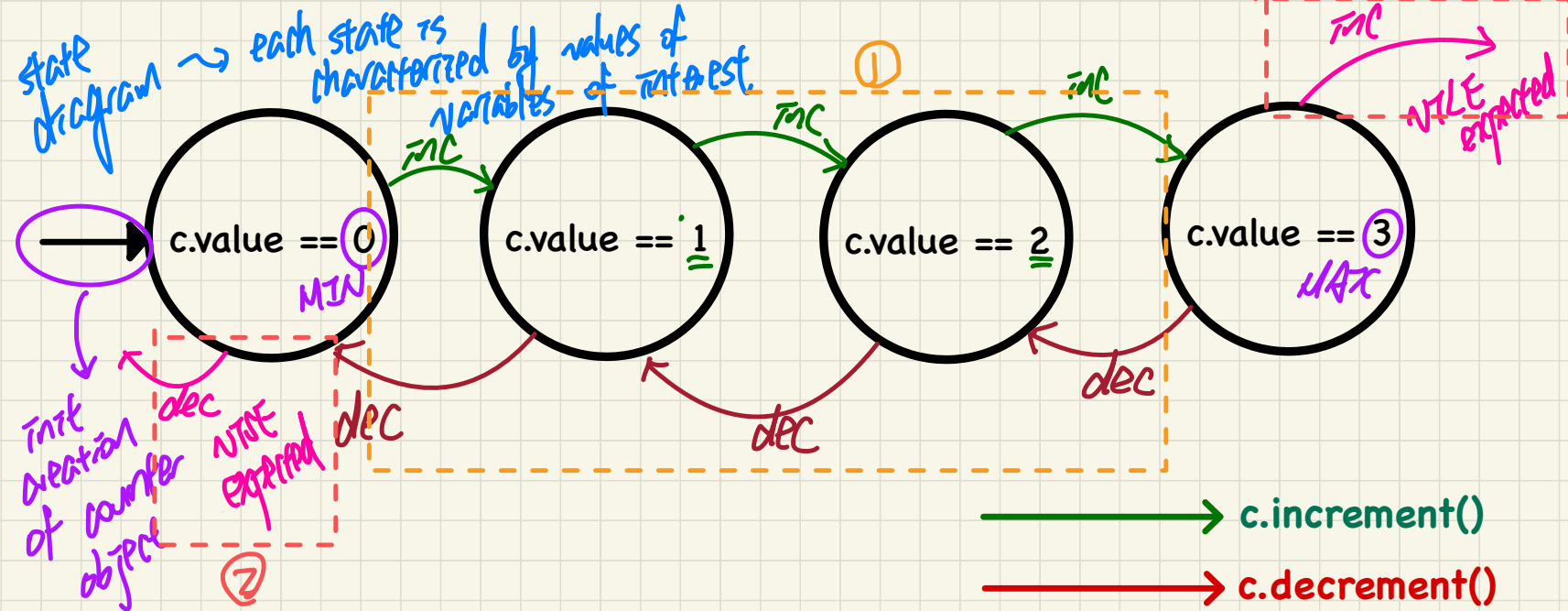
② abnormal scenarios to test

Coming Up with Test Cases: A Single, Bounded Variable

② ~~if~~ exception occurred \rightarrow fail; exception not occurred \rightarrow pass

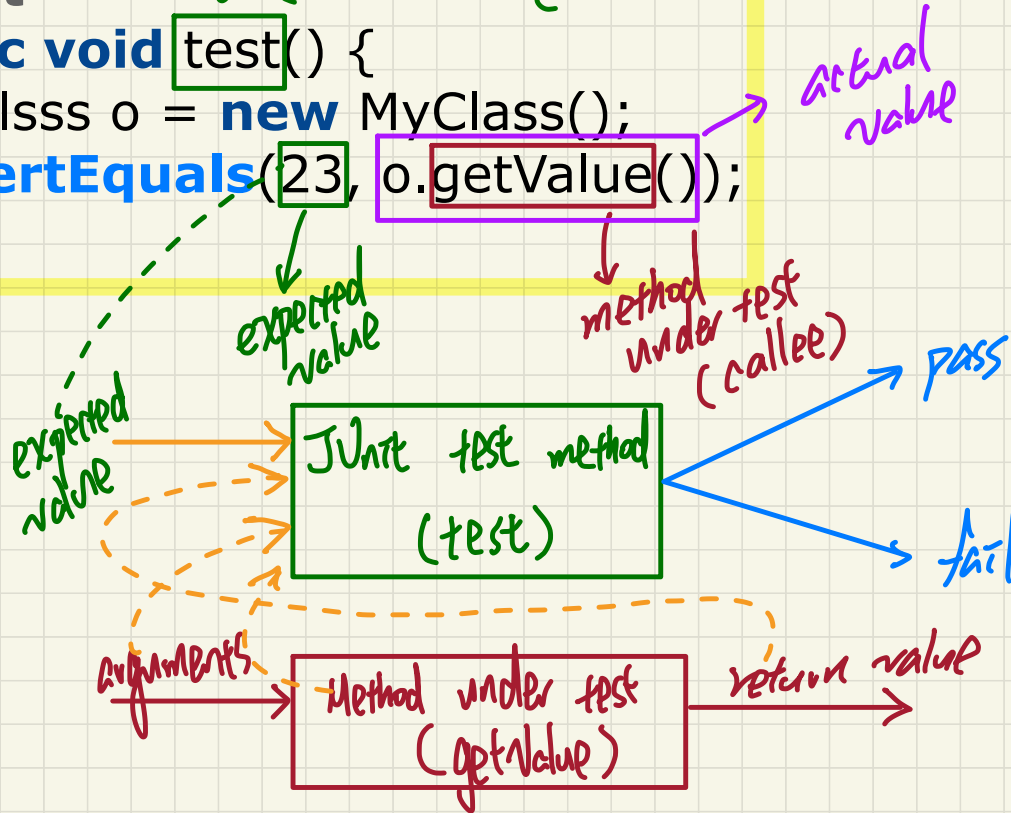
Boundries: ② exception occurred \rightarrow pass not occurred \rightarrow fail

```
Counter.MIN_VALUE <= c.value <= Counter.MAX_VALUE
```

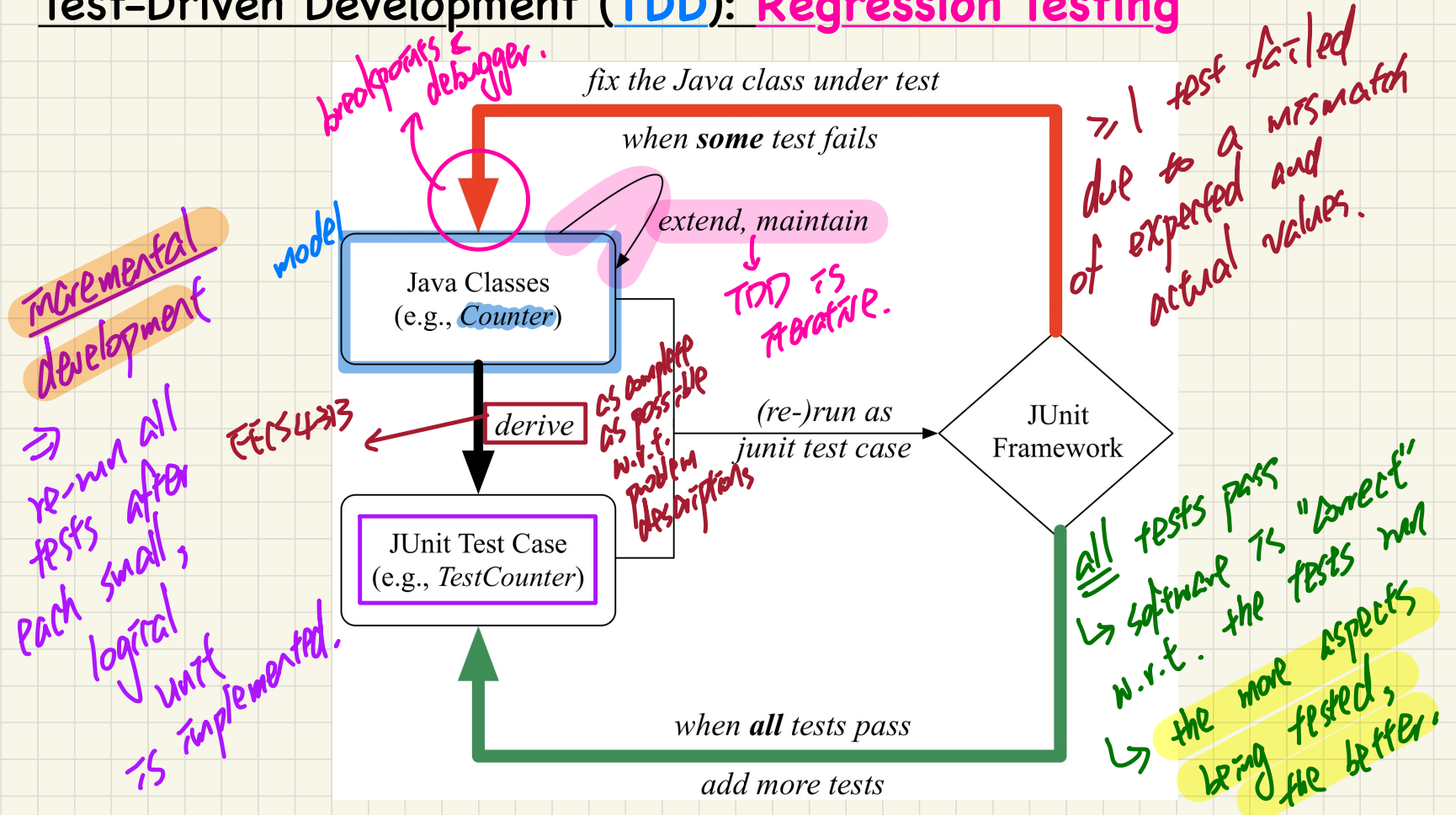


JUnit Test Method vs. Method Under Test

```
@Test
public void test() {
    MyClass o = new MyClass();
    assertEquals(23, o.getValue());
}
```



Test-Driven Development (TDD): Regression Testing



A Default Test Case that **Fails**

slide 16

The result of running a test is considered:

- **Failure** if either
 - an **assertion failure** (e.g., caused by `fail`, `assertTrue`, `assertEquals`) occurs
 - an unexpected **exception** (e.g., `NullPointerException`, `ArrayIndexOutOfBoundsException`) thrown
- **Success** if neither **assertion failures** nor (unexpected) **exceptions** occur.

slide 19

leaving the body
of test method
empty
will pass the test
initially

```
TestCounter.java
1 package tests;
2 import static org.junit.Assert.*;
3 import org.junit.Test;
4 public class TestCounter {
5     @Test
6     public void test() {
7         fail("Not yet implemented");
8     }
9 }
10
```

✓ Q: What is the easiest way to making this test **pass**?

Examples: JUnit **Assertions** (1)

std 23

Consider the following class:

```
public class Point {  
    private int x; private int y;  
    public Point(int x, int y) { this.x = x; this.y = y; }  
    public int getX() { return this.x; }  
    public int getY() { return this.y; }  
}
```

Then consider these assertions. Do they **pass** or **fail**?

```
Point p;  
assertNull(p);  
assertTrue(p == null);  
assertFalse(p != null);  
assertEquals(3, p.getX());  
p = new Point(3, 4);  
assertNull(p);  
assertTrue(p == null);  
assertFalse(p != null);  
assertEquals(3, p.getX());  
assertTrue(p.getX() == 3 && p.getY() == 4);
```

Examples: JUnit Assertions (2)

slide 24

Consider the following class:

```
class Circle {  
    double radius;  
    Circle(double radius) { this.radius = radius; }  
    int getArea() { return 3.14 * radius * radius; }  
}
```

Then consider these assertions. Do they *pass* or *fail*?

```
Circle c = new Circle(3.4);  
assertEquals(36.2984, c.getArea(), 0.01);
```

Σ



Test

- ↳ Expectation: certain exception occurs
 - ↳ that kind of exception occurs → pass
 - ↳ that kind of exception does not occur → fail
- ↳ Expectation: certain exception does not occur
 - ↳ that kind of exception occurs → fail
 - ↳ that kind of exception does not occur → pass

JUnit: An Exception Not Expected

```
1 @Test
2 public void testIncAfterCreation() {
3     Counter c = new Counter();
4     assertEquals(Counter.MIN_VALUE, c.getValue());
5     try {
6         c.increment();
7         assertEquals(1, c.getValue());
8     }
9     catch (ValueTooLargeException e) {
10         /* Exception is not expected to be thrown. */
11         fail("ValueTooLargeException is not expected.");
12     }
13 }
```

may throw VTLE
VTLE did not occur → further check if value is int. to 1

What if increment is implemented correctly?

Expected Behaviour:

Calling `c.increment()`
when `c.value` is 0 should not
trigger a `ValueTooLargeException`

not expected to happen

```
1 @Test
2 public void testIncAfterCreation() {
3     Counter c = new Counter();
4     assertEquals(Counter.MIN_VALUE, c.getValue());
5     try {
6         c.increment();
7         assertEquals(1, c.getValue());
8     }
9     catch (ValueTooLargeException e) {
10         /* Exception is not expected to be thrown. */
11         fail("ValueTooLargeException is not expected.");
12     }
13 }
```

VTLE occurred unexpectedly → fail the test

What if increment is implemented incorrectly?

e.g., It throws VTLE when
`c.value < Counter.MAX_VALUE`

Running JUnit Test 1 on Correct Implementation

```
public void increment() throws ValueTooLargeException {  
    ⑤ if (value == Counter.MAX_VALUE) {  
        X throw new ValueTooLargeException("counter value is " + value);  
    } ⑥  
    → else { value++; }  
}
```

1. v == 0
0 → ①

```
1  @Test  
2  public void testIncAfterCreation() {  
3  ① Counter c = new Counter(); | c.v == 0  
4  ② assertEquals(Counter.MIN_VALUE, c.getValue()); → pass  
5  ③ try {  
6      ④ c.increment();  
7      ⑦ assertEquals(1, c.getValue()); → pass  
8  }  
9  X catch (ValueTooLargeException e) {  
10     /* Exception is not expected to be thrown. */  
11     fail("ValueTooLargeException is not expected.");  
12  
13 } ⑧ → test pass 'i no unexpected exception occurred.
```

Comment: ① no unexpected VTLCE
⑦ value of counter inc.

Running JUnit Test 1 on Incorrect Implementation

```
public void increment() throws ValueTooLargeException {  
    ⑤ if (value <= Counter.MAX_VALUE) {  
        ⑥ throw new ValueTooLargeException("counter value is " + value);  
    }  
    X else { value++; }  
}
```

NTE is unexpectedly
thrown by this incorrect
imp.

```
1  @Test  
2  public void testIncAfterCreation() {  
3      ① Counter c = new Counter(); | c.v == 0  
4      ② assertEquals(Counter.MIN_VALUE, c.getValue());  
5      ③ try {  
6          ④ c.increment();  
7          X assertEquals(1, c.getValue());  
8      }  
9      ⑦ catch (ValueTooLargeException e) {  
10         /* Exception is not expected to be thrown. */  
11         ⑧ fail("ValueTooLargeException is not expected.");  
12     }  
13 }
```

NTE thrown
unexpectedly

X → rest of the
test is bypassed. fail the test immediately

JUnit: An Exception Expected

```
1 @Test
2 public void testDecFromMinValue() {
3     Counter c = new Counter(); | c.v==0
4     assertEquals(Counter.MIN_VALUE, c.getValue());
5     try {
6         c.decrement();
7     } X fail ("ValueTooSmallException is expected.");
8     }
9     catch (ValueTooSmallException e) {
10         /* Exception is expected to be thrown. */
11     }
12 }
```

VTSE occurs as expected.

What if decrement is implemented correctly?

Expected Behaviour:

Calling `c.decrement()` when `c.value` is 0 should trigger a `ValueTooSmallException`.

```
1 @Test
2 public void testDecFromMinValue() {
3     Counter c = new Counter();
4     assertEquals(Counter.MIN_VALUE, c.getValue());
5     try {
6         c.decrement();
7     } fail ("ValueTooSmallException is expected.");
8     }
9     catch (ValueTooSmallException e) {
10         /* Exception is expected to be thrown. */
11     }
12 }
```

↳ the expected VTSE did not occur.

What if decrement is implemented incorrectly?

e.g., It only throws VTSE when `c.value < Counter.MIN_VALUE`

Running JUnit Test 2 on **Correct** Implementation

```
public void decrement() throws ValueTooSmallException {  
    ⑤ if (value == Counter.MIN VALUE) {  
        ⑥ throw new ValueTooSmallException("counter value is " + value);  
    }  
    X else { value --; }  
}
```

```
1  @Test  
2  public void testDecFromMinValue() {  
3      ① Counter c = new Counter(); | C.V == 0  
4      ② assertEquals(Counter.MIN VALUE, c.getValue());  
5      ③ try {  
6          ④ c.decrement();  
7          X fail("ValueTooSmallException is expected.");  
8      }  
9      ⑦ catch (ValueTooSmallException e) {  
10         /* Exception is expected to be thrown. */  
11     }  
12 }
```

NOTE: error as expected

Running JUnit Test 2 on **Incorrect** Implementation



```
public void decrement() throws ValueTooSmallException {  
    ⑤ if (value <= Counter.MIN_VALUE) {  
        X throw new ValueTooSmallException("counter value is " + value);  
    }  
    ⑥ else { value --; }  
}  
      0 → -1
```

```
1  @Test  
2  public void testDecFromMinValue() {  
3      ① Counter c = new Counter(); | C.V == 0  
4      ② assertEquals(Counter.MIN_VALUE, c.getValue());  
5      ③ try {  
6          ④ c.decrement();  
7          ⑦ fail("ValueTooSmallException is expected.");  
8      }  
9      X catch (ValueTooSmallException e) {  
10         /* Exception is expected to be thrown. */  
11     }  
12 }
```

→ NPE unexpectedly did not occur

X

JUnit: Exception Sometimes Expected, Sometimes Not

```
1 @Test
2 public void testIncFromMaxValue() {
3     Counter c = new Counter();
4     try {
5         c.increment(); c.increment(); c.increment();
6     }
7     catch (ValueTooLargeException e) {
8         fail("ValueTooLargeException was thrown unexpectedly.");
9     }
10    assertEquals(Counter.MAX_VALUE, c.getValue());
11    try {
12        c.increment();
13        fail("ValueTooLargeException was NOT thrown as expected.");
14    }
15    catch (ValueTooLargeException e) {
16        // Do nothing: ValueTooLargeException thrown as expected. */
17    }
18 }
```

either increment throwing VTLException is unexpected.

VTLException occurs unexpectedly

VTLException occurs as expected.

Expected Behaviour:

Calling `c.increment()`
3 times to reach `c`'s max **should not**
trigger any `ValueTooLargeException`.

Calling `c.increment()`
when `c` is already at its max **should**
trigger a `ValueTooLargeException`

→ VTLException not expected

→ VTLException expected

Running JUnit Test 3 on Correct Implementation

```
public void increment() throws ValueTooLargeException {  
    ④ if (value == Counter.MAX_VALUE) {  
        X throw new ValueTooLargeException("counter value is " + value);  
    }  
    ⑤ else { value ++; }  
    }  
    0 → 1
```

PASS -

```
1 @Test  
2 public void testIncFromMaxValue() {  
3     ① Counter c = new Counter(); | C.V == 0  
4     ② try {  
5         ③ c.increment(); c.increment(); c.increment(); | C.V == 3  
6         | 0 → 1 | 1 → 2 | 2 → 3  
7     } catch (ValueTooLargeException e) {  
8         fail("ValueTooLargeException was thrown unexpectedly.");  
9     }  
10    ④ assertEquals(Counter.MAX_VALUE, c.getValue());  
11    ⑤ try {  
12        ⑥ c.increment();  
13        X fail("ValueTooLargeException was NOT thrown as expected.");  
14    } catch (ValueTooLargeException e) {  
15        → Do nothing: ValueTooLargeException thrown as expected. */  
16    }  
17    }  
18    }  
    ⑦  
    ⑧  
    ⑨  
    ⑩  
    ⑪  
    ⑫  
    ⑬  
    ⑭  
    ⑮  
    ⑯  
    ⑰  
    ⑱  
    ⑲  
    ⑳  
    ㉑  
    ㉒  
    ㉓  
    ㉔  
    ㉕  
    ㉖  
    ㉗  
    ㉘  
    ㉙  
    ㉚  
    ㉛  
    ㉜  
    ㉝  
    ㉞  
    ㉟  
    ㊱  
    ㊲  
    ㊳  
    ㊴  
    ㊵  
    ㊶  
    ㊷  
    ㊸  
    ㊹  
    ㊺  
    ㊻  
    ㊼  
    ㊽  
    ㊾  
    ㊿  
    ㏀  
    ㏁  
    ㏂  
    ㏃  
    ㏄  
    ㏅  
    ㏆  
    ㏇  
    ㏈  
    ㏉  
    ㏊  
    ㏋  
    ㏌  
    ㏍  
    ㏎  
    ㏏  
    ㏐  
    ㏑  
    ㏒  
    ㏓  
    ㏔  
    ㏕  
    ㏖  
    ㏗  
    ㏘  
    ㏙  
    ㏚  
    ㏛  
    ㏜  
    ㏝  
    ㏞  
    ㏟  
    ㏠  
    ㏡  
    ㏢  
    ㏣  
    ㏤  
    ㏥  
    ㏦  
    ㏧  
    ㏨  
    ㏩  
    ㏪  
    ㏫  
    ㏬  
    ㏭  
    ㏮  
    ㏯  
    ㏰  
    ㏱  
    ㏲  
    ㏳  
    ㏴  
    ㏵  
    ㏶  
    ㏷  
    ㏸  
    ㏹  
    ㏺  
    ㏻  
    ㏼  
    ㏽  
    ㏾  
    ㏿  
    㐀  
    㐁  
    㐂  
    㐃  
    㐄  
    㐅  
    㐆  
    㐇  
    㐈  
    㐉  
    㐊  
    㐋  
    㐌  
    㐍  
    㐎  
    㐏  
    㐐  
    㐑  
    㐒  
    㐓  
    㐔  
    㐕  
    㐖  
    㐗  
    㐘  
    㐙  
    㐚  
    㐛  
    㐜  
    㐝  
    㐞  
    㐟  
    㐠  
    㐡  
    㐢  
    㐣  
    㐤  
    㐥  
    㐦  
    㐧  
    㐨  
    㐩  
    㐪  
    㐫  
    㐬  
    㐭  
    㐮  
    㐯  
    㐰  
    㐱  
    㐲  
    㐳  
    㐴  
    㐵  
    㐶  
    㐷  
    㐸  
    㐹  
    㐺  
    㐻  
    㐼  
    㐽  
    㐾  
    㐿  
    㑀  
    㑁  
    㑂  
    㑃  
    㑄  
    㑅  
    㑆  
    㑇  
    㑈  
    㑉  
    㑊  
    㑋  
    㑌  
    㑍  
    㑎  
    㑏  
    㑐  
    㑑  
    㑒  
    㑓  
    㑔  
    㑕  
    㑖  
    㑗  
    㑘  
    㑙  
    㑚  
    㑛  
    㑜  
    㑝  
    㑞  
    㑟  
    㑠  
    㑡  
    㑢  
    㑣  
    㑤  
    㑥  
    㑦  
    㑧  
    㑨  
    㑩  
    㑪  
    㑫  
    㑬  
    㑭  
    㑮  
    㑯  
    㑰  
    㑱  
    㑲  
    㑳  
    㑴  
    㑵  
    㑶  
    㑷  
    㑸  
    㑹  
    㑺  
    㑻  
    㑼  
    㑽  
    㑾  
    㑿  
    㒀  
    㒁  
    㒂  
    㒃  
    㒄  
    㒅  
    㒆  
    㒇  
    㒈  
    㒉  
    㒊  
    㒋  
    㒌  
    㒍  
    㒎  
    㒏  
    㒐  
    㒑  
    㒒  
    㒓  
    㒔  
    㒕  
    㒖  
    㒗  
    㒘  
    㒙  
    㒚  
    㒛  
    㒜  
    㒝  
    㒞  
    㒟  
    㒠  
    㒡  
    㒢  
    㒣  
    㒤  
    㒥  
    㒦  
    㒧  
    㒨  
    㒩  
    㒪  
    㒫  
    㒬  
    㒭  
    㒮  
    㒯  
    㒰  
    㒱  
    㒲  
    㒳  
    㒴  
    㒵  
    㒶  
    㒷  
    㒸  
    㒹  
    㒺  
    㒻  
    㒼  
    㒽  
    㒾  
    㒿  
    㓀  
    㓁  
    㓂  
    㓃  
    㓄  
    㓅  
    㓆  
    㓇  
    㓈  
    㓉  
    㓊  
    㓋  
    㓌  
    㓍  
    㓎  
    㓏  
    㓐  
    㓑  
    㓒  
    㓓  
    㓔  
    㓕  
    㓖  
    㓗  
    㓘  
    㓙  
    㓚  
    㓛  
    㓜  
    㓝  
    㓞  
    㓟  
    㓠  
    㓡  
    㓢  
    㓣  
    㓤  
    㓥  
    㓦  
    㓧  
    㓨  
    㓩  
    㓪  
    㓫  
    㓬  
    㓭  
    㓮  
    㓯  
    㓰  
    㓱  
    㓲  
    㓳  
    㓴  
    㓵  
    㓶  
    㓷  
    㓸  
    㓹  
    㓺  
    㓻  
    㓼  
    㓽  
    㓾  
    㓿  
    㔀  
    㔁  
    㔂  
    㔃  
    㔄  
    㔅  
    㔆  
    㔇  
    㔈  
    㔉  
    㔊  
    㔋  
    㔌  
    㔍  
    㔎  
    㔏  
    㔐  
    㔑  
    㔒  
    㔓  
    㔔  
    㔕  
    㔖  
    㔗  
    㔘  
    㔙  
    㔚  
    㔛  
    㔜  
    㔝  
    㔞  
    㔟  
    㔠  
    㔡  
    㔢  
    㔣  
    㔤  
    㔥  
    㔦  
    㔧  
    㔨  
    㔩  
    㔪  
    㔫  
    㔬  
    㔭  
    㔮  
    㔯  
    㔰  
    㔱  
    㔲  
    㔳  
    㔴  
    㔵  
    㔶  
    㔷  
    㔸  
    㔹  
    㔺  
    㔻  
    㔼  
    㔽  
    㔾  
    㔿  
    㕀  
    㕁  
    㕂  
    㕃  
    㕄  
    㕅  
    㕆  
    㕇  
    㕈  
    㕉  
    㕊  
    㕋  
    㕌  
    㕍  
    㕎  
    㕏  
    㕐  
    㕑  
    㕒  
    㕓  
    㕔  
    㕕  
    㕖  
    㕗  
    㕘  
    㕙  
    㕚  
    㕛  
    㕜  
    㕝  
    㕞  
    㕟  
    㕠  
    㕡  
    㕢  
    㕣  
    㕤  
    㕥  
    㕦  
    㕧  
    㕨  
    㕩  
    㕪  
    㕫  
    㕬  
    㕭  
    㕮  
    㕯  
    㕰  
    㕱  
    㕲  
    㕳  
    㕴  
    㕵  
    㕶  
    㕷  
    㕸  
    㕹  
    㕺  
    㕻  
    㕼  
    㕽  
    㕾  
    㕿  
    㖀  
    㖁  
    㖂  
    㖃  
    㖄  
    㖅  
    㖆  
    㖇  
    㖈  
    㖉  
    㖊  
    㖋  
    㖌  
    㖍  
    㖎  
    㖏  
    㖐  
    㖑  
    㖒  
    㖓  
    㖔  
    㖕  
    㖖  
    㖗  
    㖘  
    㖙  
    㖚  
    㖛  
    㖜  
    㖝  
    㖞  
    㖟  
    㖠  
    㖡  
    㖢  
    㖣  
    㖤  
    㖥  
    㖦  
    㖧  
    㖨  
    㖩  
    㖪  
    㖫  
    㖬  
    㖭  
    㖮  
    㖯  
    㖰  
    㖱  
    㖲  
    㖳  
    㖴  
    㖵  
    㖶  
    㖷  
    㖸  
    㖹  
    㖺  
    㖻  
    㖼  
    㖽  
    㖾  
    㖿  
    㗀  
    㗁  
    㗂  
    㗃  
    㗄  
    㗅  
    㗆  
    㗇  
    㗈  
    㗉  
    㗊  
    㗋  
    㗌  
    㗍  
    㗎  
    㗏  
    㗐  
    㗑  
    㗒  
    㗓  
    㗔  
    㗕  
    㗖  
    㗗  
    㗘  
    㗙  
    㗚  
    㗛  
    㗜  
    㗝  
    㗞  
    㗟  
    㗠  
    㗡  
    㗢  
    㗣  
    㗤  
    㗥  
    㗦  
    㗧  
    㗨  
    㗩  
    㗪  
    㗫  
    㗬  
    㗭  
    㗮  
    㗯  
    㗰  
    㗱  
    㗲  
    㗳  
    㗴  
    㗵  
    㗶  
    㗷  
    㗸  
    㗹  
    㗺  
    㗻  
    㗼  
    㗽  
    㗾  
    㗿  
    㘀  
    㘁  
    㘂  
    㘃  
    㘄  
    㘅  
    㘆  
    㘇  
    㘈  
    㘉  
    㘊  
    㘋  
    㘌  
    㘍  
    㘎  
    㘏  
    㘐  
    㘑  
    㘒  
    㘓  
    㘔  
    㘕  
    㘖  
    㘗  
    㘘  
    㘙  
    㘚  
    㘛  
    㘜  
    㘝  
    㘞  
    㘟  
    㘠  
    㘡  
    㘢  
    㘣  
    㘤  
    㘥  
    㘦  
    㘧  
    㘨  
    㘩  
    㘪  
    㘫  
    㘬  
    㘭  
    㘮  
    㘯  
    㘰  
    㘱  
    㘲  
    㘳  
    㘴  
    㘵  
    㘶  
    㘷  
    㘸  
    㘹  
    㘺  
    㘻  
    㘼  
    㘽  
    㘾  
    㘿  
    㙀  
    㙁  
    㙂  
    㙃  
    㙄  
    㙅  
    㙆  
    㙇  
    㙈  
    㙉  
    㙊  
    㙋  
    㙌  
    㙍  
    㙎  
    㙏  
    㙐  
    㙑  
    㙒  
    㙓  
    㙔  
    㙕  
    㙖  
    㙗  
    㙘  
    㙙  
    㙚  
    㙛  
    㙜  
    㙝  
    㙞  
    㙟  
    㙠  
    㙡  
    㙢  
    㙣  
    㙤  
    㙥  
    㙦  
    㙧  
    㙨  
    㙩  
    㙪  
    㙫  
    㙬  
    㙭  
    㙮  
    㙯  
    㙰  
    㙱  
    㙲  
    㙳  
    㙴  
    㙵  
    㙶  
    㙷  
    㙸  
    㙹  
    㙺  
    㙻  
    㙼  
    㙽  
    㙾  
    㙿  
    㚀  
    㚁  
    㚂  
    㚃  
    㚄  
    㚅  
    㚆  
    㚇  
    㚈  
    㚉  
    㚊  
    㚋  
    㚌  
    㚍  
    㚎  
    㚏  
    㚐  
    㚑  
    㚒  
    㚓  
    㚔  
    㚕  
    㚖  
    㚗  
    㚘  
    㚙  
    㚚  
    㚛  
    㚜  
    㚝  
    㚞  
    㚟  
    㚠  
    㚡  
    㚢  
    㚣  
    㚤  
    㚥  
    㚦  
    㚧  
    㚨  
    㚩  
    㚪  
    㚫  
    㚬  
    㚭  
    㚮  
    㚯  
    㚰  
    㚱  
    㚲  
    㚳  
    㚴  
    㚵  
    㚶  
    㚷  
    㚸  
    㚹  
    㚺  
    㚻  
    㚼  
    㚽  
    㚾  
    㚿  
    㞀  
    㞁  
    㞂  
    㞃  
    㞄  
    㞅  
    㞆  
    㞇  
    㞈  
    㞉  
    㞊  
    㞋  
    㞌  
    㞍  
    㞎  
    㞏  
    㞐  
    㞑  
    㞒  
    㞓  
    㞔  
    㞕  
    㞖  
    㞗  
    㞘  
    㞙  
    㞚  
    㞛  
    㞜  
    㞝  
    㞞  
    㞟  
    㞠  
    㞡  
    㞢  
    㞣  
    㞤  
    㞥  
    㞦  
    㞧  
    㞨  
    㞩  
    㞪  
    㞫  
    㞬  
    㞭  
    㞮  
    㞯  
    㞰  
    㞱  
    㞲  
    㞳  
    㞴  
    㞵  
    㞶  
    㞷  
    㞸  
    㞹  
    㞺  
    㞻  
    㞼  
    㞽  
    㞾  
    㞿  
    㟀  
    㟁  
    㟂  
    㟃  
    㟄  
    㟅  
    㟆  
    㟇  
    㟈  
    㟉  
    㟊  
    㟋  
    㟌  
    㟍  
    㟎  
    㟏  
    㟐  
    㟑  
    㟒  
    㟓  
    㟔  
    㟕  
    㟖  
    㟗  
    㟘  
    㟙  
    㟚  
    㟛  
    㟜  
    㟝  
    㟞  
    㟟  
    㟠  
    㟡  
    㟢  
    㟣  
    㟤  
    㟥  
    㟦  
    㟧  
    㟨  
    㟩  
    㟪  
    㟫  
    㟬  
    㟭  
    㟮  
    㟯  
    㟰  
    㟱  
    㟲  
    㟳  
    㟴  
    㟵  
    㟶  
    㟷  
    㟸  
    㟹  
    㟺  
    㟻  
    㟼  
    㟽  
    㟾  
    㟿  
    㠀  
    㠁  
    㠂  
    㠃  
    㠄  
    㠅  
    㠆  
    㠇  
    㠈  
    㠉  
    㠊  
    㠋  
    㠌  
    㠍  
    㠎  
    㠏  
    㠐  
    㠑  
    㠒  
    㠓  
    㠔  
    㠕  
    㠖  
    㠗  
    㠘  
    㠙  
    㠚  
    㠛  
    㠜  
    㠝  
    㠞  
    㠟  
    㠠  
    㠡  
    㠢  
    㠣  
    㠤  
    㠥  
    㠦  
    㠧  
    㠨  
    㠩  
    㠪  
    㠫  
    㠬  
    㠭  
    㠮  
    㠯  
    㠰  
    㠱  
    㠲  
    㠳  
    㠴  
    㠵  
    㠶  
    㠷  
    㠸  
    㠹  
    㠺  
    㠻  
    㠼  
    㠽  
    㠾  
    㠿  
    㡀  
    㡁  
    㡂  
    㡃  
    㡄  
    㡅  
    㡆  
    㡇  
    㡈  
    㡉  
    㡊  
    㡋  
    㡌  
    㡍  
    㡎  
    㡏  
    㡐  
    㡑  
    㡒  
    㡓  
    㡔  
    㡕  
    㡖  
    㡗  
    㡘  
    㡙  
    㡚  
    㡛  
    㡜  
    㡝  
    㡞  
    㡟  
    㡠  
    㡡  
    㡢  
    㡣  
    㡤  
    㡥  
    㡦  
    㡧  
    㡨  
    㡩  
    㡪  
    㡫  
    㡬  
    㡭  
    㡮  
    㡯  
    㡰  
    㡱  
    㡲  
    㡳  
    㡴  
    㡵  
    㡶  
    㡷  
    㡸  
    㡹  
    㡺  
    㡻  
    㡼  
    㡽  
    㡾  
    㡿  
    㢀  
    㢁  
    㢂  
    㢃  
    㢄  
    㢅  
    㢆  
    㢇  
    㢈  
    㢉  
    㢊  
    㢋  
    㢌  
    㢍  
    㢎  
    㢏  
    㢐  
    㢑  
    㢒  
    㢓  
    㢔  
    㢕  
    㢖  
    㢗  
    㢘  
    㢙  
    㢚  
    㢛  
    㢜  
    㢝  
    㢞  
    㢟  
    㢠  
    㢡  
    㢢  
    㢣  
    㢤  
    㢥  
    㢦  
    㢧  
    㢨  
    㢩  
    㢪  
    㢫  
    㢬  
    㢭  
    㢮  
    㢯  
    㢰  
    㢱  
    㢲  
    㢳  
    㢴  
    㢵  
    㢶  
    㢷  
    㢸  
    㢹  
    㢺  
    㢻  
    㢼  
    㢽  
    㢾  
    㢿  
    㣀  
    㣁  
    㣂  
    㣃  
    㣄  
    㣅  
    㣆  
    㣇  
    㣈  
    㣉  
    㣊  
    㣋  
    㣌  
    㣍  
    㣎  
    㣏  
    㣐  
    㣑  
    㣒  
    㣓  
    㣔  
    㣕  
    㣖  
    㣗  
    㣘  
    㣙  
    㣚  
    㣛  
    㣜  
    㣝  
    㣞  
    㣟  
    㣠  
    㣡  
    㣢  
    㣣  
    㣤  
    㣥  
    㣦  
    㣧  
    㣨  
    㣩  
    㣪  
    㣫  
    㣬  
    㣭  
    㣮  
    㣯  
    㣰  
    㣱  
    㣲  
    㣳  
    㣴  
    㣵  
    㣶  
    㣷  
    㣸  
    㣹  
    㣺  
    㣻  
    㣼  
    㣽  
    㣾  
    㣿  
    㤀  
    㤁  
    㤂  
    㤃  
    㤄  
    㤅  
    㤆  
    㤇  
    㤈  
    㤉  
    㤊  
    㤋  
    㤌  
    㤍  
    㤎  
    㤏  
    㤐  
    㤑  
    㤒  
    㤓  
    㤔  
    㤕  
    㤖  
    㤗  
    㤘  
    㤙  
    㤚  
    㤛  
    㤜  
    㤝  
    㤞  
    㤟  
    㤠  
    㤡  
    㤢  
    㤣  
    㤤  
    㤥  
    㤦  
    㤧  
    㤨  
    㤩  
    㤪  
    㤫  
    㤬  
    㤭  
    㤮  
    㤯  
    㤰  
    㤱  
    㤲  
    㤳  
    㤴  
    㤵  
    㤶  
    㤷  
    㤸  
    㤹  
    㤺  
    㤻  
    㤼  
    㤽  
    㤾  
    㤿  
    㥀  
    㥁  
    㥂  
    㥃  
    㥄  
    㥅  
    㥆  
    㥇  
    㥈  
    㥉  
    㥊  
    㥋  
    㥌  
    㥍  
    㥎  
    㥏  
    㥐  
    㥑  
    㥒  
    㥓  
    㥔  
    㥕  
    㥖  
    㥗  
    㥘  
    㥙  
    㥚  
    㥛  
    㥜  
    㥝  
    㥞  
    㥟  
    㥠  
    㥡  
    㥢  
    㥣  
    㥤  
    㥥  
    㥦  
    㥧  
    㥨  
    㥩  
    㥪  
    㥫  
    㥬  
    㥭  
    㥮  
    㥯  
    㥰  
    㥱  
    㥲  
    㥳  
    㥴  
    㥵  
    㥶  
    㥷  
    㥸  
    㥹  
    㥺  
    㥻  
    㥼  
    㥽  
    㥾  
    㥿  
    㦀  
    㦁  
    㦂  
    㦃  
    㦄  
    㦅  
    㦆  
    㦇  
    㦈  
    㦉  
    㦊  
    㦋  
    㦌  
    㦍  
    㦎  
    㦏  
    㦐  
    㦑  
    㦒  
    㦓  
    㦔  
    㦕  
    㦖  
    㦗  
    㦘  
    㦙  
    㦚  
    㦛  
    㦜  
    㦝  
    㦞  
    㦟  
    㦠  
    㦡  
    㦢  
    㦣  
    㦤  
    㦥  
    㦦  
    㦧  
    㦨  
    㦩  
    㦪  
    㦫  
    㦬  
    㦭  
    㦮  
    㦯  
    㦰  
    㦱  
    㦲  
    㦳  
    㦴  
    㦵  
    㦶  
    㦷  
    㦸  
    㦹  
    㦺  
    㦻  
    㦼  
    㦽  
    㦾  
    㦿  
    㧀  
    㧁  
    㧂  
    㧃  
    㧄  
    㧅  
    㧆  
    㧇  
    㧈  
    㧉  
    㧊  
    㧋  
    㧌  
    㧍  
    㧎  
    㧏  
    㧐  
    㧑  
    㧒  
    㧓  
    㧔  
    㧕  
    㧖  
    㧗  
    㧘  
    㧙  
    㧚  
    㧛  
    㧜  
    㧝  
    㧞  
    㧟  
    㧠  
    㧡  
    㧢  
    㧣  
    㧤  
    㧥  
    㧦  
    㧧  
    㧨  
    㧩  
    㧪  
    㧫  
    㧬  
    㧭  
    㧮  
    㧯  
    㧰  
    㧱  
    㧲  
    㧳  
    㧴  
    㧵  
    㧶  
    㧷  
    㧸  
    㧹  
    㧺  
    㧻  
    㧼  
    㧽  
    㧾  
    㧿  
    㨀  
    㨁  
    㨂  
    㨃  
    㨄  
    㨅  
    㨆  
    㨇  
    㨈  
    㨉  
    㨊  
    㨋  
    㨌  
    㨍  
    㨎  
    㨏  
    㨐  
    㨑  
    㨒  
    㨓  
    㨔  
    㨕  
    㨖  
    㨗  
    㨘  
    㨙  
    㨚  
    㨛  
    㨜  
    㨝  
    㨞  
    㨟  
    㨠  
    㨡  
    㨢  
    㨣  
    㨤  
    㨥  
    㨦  
    㨧  
    㨨  
    㨩  
    㨪  
    㨫  
    㨬  
    㨭  
    㨮  
    㨯  
    㨰  
    㨱  
    㨲  
    㨳  
    㨴  
    㨵  
    㨶  
    㨷  
    㨸  
    㨹  
    㨺  
    㨻  
    㨼  
    㨽  
    㨾  
    㨿  
    㩀  
    㩁  
    㩂  
    㩃  
    㩄  
    㩅  
    㩆  
    㩇  
    㩈  
    㩉  
    㩊  
    㩋  
    㩌  
    㩍  
    㩎  
    㩏  
    㩐  
    㩑  
    㩒  
    㩓  
    㩔  
    㩕  
    㩖  
    㩗  
    㩘  
    㩙  
    㩚  
    㩛  
    㩜  
    㩝  
    㩞  
    㩟  
    㩠  
    㩡  
    㩢  
    㩣  
    㩤  
    㩥  
    㩦  
    㩧  
    㩨  
    㩩  
    㩪  
    㩫  
    㩬  
    㩭  
    㩮  
    㩯  
    㩰  
    㩱  
    㩲  
    㩳  
    㩴  
    㩵  
    㩶  
    㩷  
    㩸  
    㩹  
    㩺  
    㩻  
    㩼  
    㩽  
    㩾  
    㩿  
    㪀  
    㪁  
    㪂  
    㪃  
    㪄  
    㪅  
    㪆  
    㪇  
    㪈  
    㪉  
    㪊  
    㪋  
    㪌  
    㪍  
    㪎  
    㪏  
    㪐  
    㪑  
    㪒  
    㪓  
    㪔  
    㪕  
    㪖  
    㪗  
    㪘  
    㪙  
    㪚  
    㪛  
    㪜  
    㪝  
    㪞  
    㪟  
    㪠  
    㪡  
    㪢  
    㪣  
    㪤  
    㪥  
    㪦  
    㪧  
    㪨  
    㪩  
    㪪  
    㪫  
    㪬  
    㪭  
    㪮  
    㪯  
    㪰  
    㪱  
    㪲  
    㪳  
    㪴  
    㪵  
    㪶  
    㪷  
    㪸  
    㪹  
    㪺  
    㪻  
    㪼  
    㪽  
    㪾  
    㪿  
    㫀  
    㫁  
    㫂  
    㫃  
    㫄  
    㫅  
    㫆  
    㫇  
    㫈  
    㫉  
    㫊  
    㫋  
    㫌  
    㫍  
    㫎  
    㫏  
    㫐  
    㫑  
    㫒  
    㫓  
    㫔  
    㫕  
    㫖  
    㫗  
    㫘  
    㫙  
    㫚  
    㫛  
    㫜  
    㫝  
    㫞  
    㫟  
    㫠  
    㫡  
    㫢  
    㫣  
    㫤  
    㫥  
    㫦  
    㫧  
    㫨  
    㫩  
    㫪  
    㫫  
    㫬  
    㫭  
    㫮  
    㫯  
    㫰  
    㫱  
    㫲  
    㫳  
    㫴  
    㫵  
    㫶  
    㫷  
    㫸  
    
```