Started on	Tuesday, 9 February 2021, 2:19 PM
State	Finished
Completed on	Tuesday, 9 February 2021, 2:21 PM
Time taken	2 mins 2 secs
Marks	10.00/10.00
Grade	**100.00** out of 100.00

Question **1**

Correct

Mark 1.00 out of 1.00

A CPU generates 32-bit virtual addresses. The page size is 4 KB. The processor has a translation look-aside buffer (TLB) which can hold a total of 128 page table entries and is 4-way set associative. The minimum size of the TLB tag is:

Select one:

- **a. 20 bits**
- **b. 13 bits**
- **c. 11 bits**
- **d. 15 bits** ✓ Size of a page = 4KB = 2^12 Total number of bits needed to address a page frame = 32 – 12 = 20 If there are 'n' cache lines in a set, the cache placement is called n-way set associative. Since TLB is 4 way set associative and can hold total 128 (2^7) page table entries, number of sets in cache = 2^7/4 = 2^5. So 5 bits are needed to address a set, and 15 (20 – 5) bits are needed for tag.

The correct answer is: 15 bits

Question 2

Correct

Mark 1.00 out of 1.00

Consider a system with byte-addressable memory, 32 bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. The size of the page table in the system in megabytes is _____

Select one:

- **a. 16**
- **b. 2**
- **c. 4** ✓ Number of entries in page table = 232 / 4Kbyte
  = 232 / 212
  = 220

  Size of page table = (No. page table entries)*(Size of an entry)
  = 220 * 4 bytes
  = 222
  = 4 Megabytes
- **d. 8**

The correct answer is: 4

Question 3

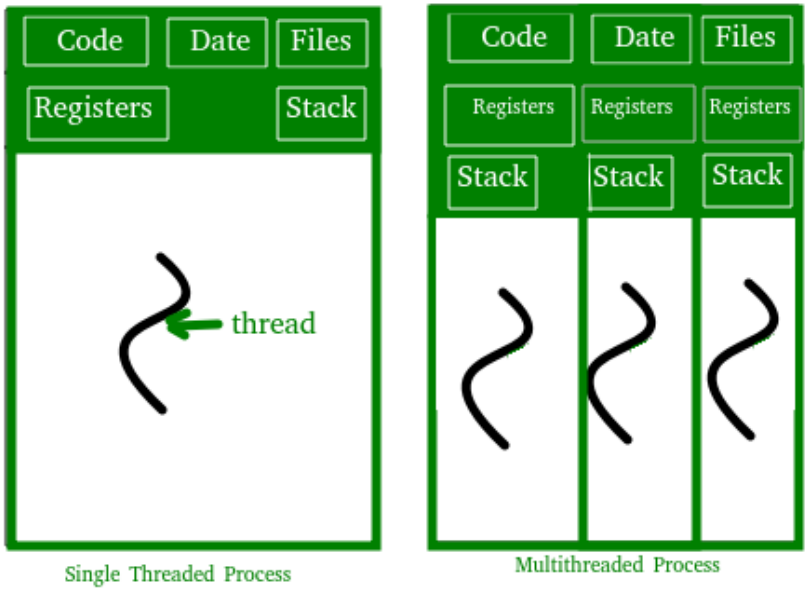
Correct

Mark 1.00 out of 1.00

Which one of the following is NOT shared by the threads of the same process?

Select one:

- **a. File Descriptor Table**
- **b. Stack** ✓ Threads can not share stack (used for maintaining function calls) as they may have their individual function call sequence.



- **c. Address Space**
- **d. Message Queue**

The correct answer is: Stack

Question 4

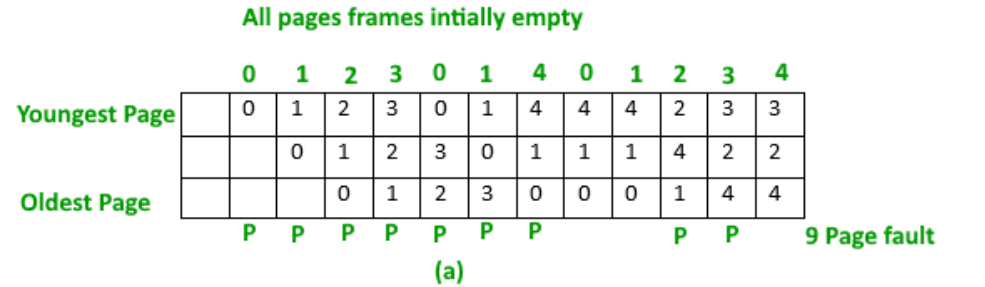
Correct

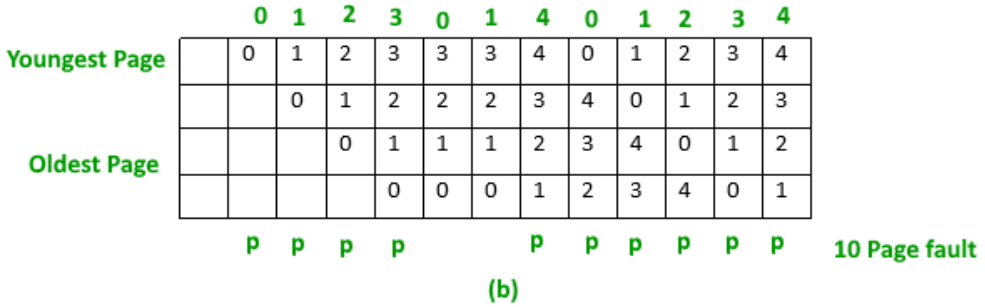
Mark 1.00 out of 1.00

In which one of the following page replacement algorithms it is possible for the page fault rate to increase even when the number of allocated frames increases?

Select one:

- **a. MRU (Most Recently Used)**
- **b. OPT (Optimal Page Replacement)**
- **c. LRU (Least Recently Used)**
- **d. FIFO (First In First Out)** ✓ In some situations FIFO page replacement gives more page faults when increasing the number of page frames. This situation is Belady's anomaly. Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3 2 1 0 3 2 4 3 2 1 0 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.





The correct answer is: FIFO (First In First Out)

Question 5

Correct

Mark 1.00 out of 1.00

What is the result of the following?

```
1: public class Order {
2: static String result = "";
3: { result += "c"; }
4: static
5: { result += "u"; }
6: { result += "r"; }
7: }
```

```
1: public class OrderDriver {
2: public static void main(String[] args) {
3: System.out.print(Order.result + " ");
4: System.out.print(Order.result + " ");
5: new Order();
6: new Order();
7: System.out.print(Order.result + " ");
8: }
9: }
```

Select one or more:

- **a. curur**
- **b. ucrcr**
- **c. u ucrcr**
- **d. u u curcur**
- **e. u u ucrcr** ✓

  On line 3 of OrderDriver, we refer to Order for the first time. At this point the statics in Order get initialized. In this case, the statics are the static declaration of result and the static initializer. result is u at this point. On line 4, result is the same because the static initialization is only run once. On line 5, we create a new Order, which triggers the instance initializers in the order they appear in the file. Now result is ucr. Line 6 creates another Order, triggering another set of initializers. Now result is ucrcr. Notice how the static is on a different line than the initialization code in lines 4–5 of Order. The exam may try to trick you by formatting the code like this to confuse you.

- **f. ur ur urc**

g. The code does not compile

Your answer is correct.

The correct answer is: u u ucrcr

Question 6

Correct

Mark 1.00 out of 1.00

Which of the following are true about the following code? (Choose all that apply)

```
public class Create {
Create() {
System.out.print("1 ");
}
Create(int num) {
System.out.print("2 ");
}
Create(Integer num) {
System.out.print("3 ");
}
Create(Object num) {
System.out.print("4 ");
}
Create(int... nums) {
System.out.print("5 ");
}
public static void main(String[] args) {
new Create(100);
new Create(1000L);
}
}
```

Select one or more:

- [x] **a. The code prints out 2 4.** ✓
- [ ] **b. The code prints out 3 4.**
- [ ] **c. The code prints out 4 2.**
- [ ] **d. The code prints out 4 4.**
- [x] **e. The code prints 3 4 if you remove the constructor Create(int num).** ✓

The 100 parameter is an int and so calls the matching int constructor. When this constructor is removed, Java looks for the next most specific constructor. Java prefers autoboxing to varargs, and so chooses the Integer constructor. The 100L parameter

is a long. Since it can’t be converted into a smaller type, it is autoboxed into a Long and then the constructor for Object is called.

- [ ] **f. The code prints 4 4 if you remove the constructor Create(int num).**
- [ ] **g. The code prints 5 4 if you remove the constructor Create(int num).**

Your answer is correct.

The correct answer is: The code prints out 2 4., The code prints 3 4 if you remove the constructor Create(int num).

Question 7

Correct

Mark 1.00 out of 1.00

What is the result of the following code?

```
1: interface Climb {
2: boolean isTooHigh(int height, int limit);
3: }
4:
5: public class Climber {
6: public static void main(String[] args) {
7: check((h, l) -> h.append(l).isEmpty(), 5);
8: }
9: private static void check(Climb climb, int height) {
10: if (climb.isTooHigh(height, 10))
11: System.out.println("too high");
12: else
13: System.out.println("ok");
14: }
15: }
```

Select one or more:

- **a. Compiler error on line 10.**
- **b. ok**
- **c. Compiler error on a different line.**
- **d. Compiler error on line 7.** ✓

  The interface takes two int parameters. The code on line 7 attempts to use them as if one is a StringBuilder. It is tricky to use types in a lambda when they are implicitly specified. Remember to check the interface for the real type.

- **e. too high**

Your answer is correct.

The correct answer is: Compiler error on line 7.

Question 8

Correct

Mark 1.00 out of 1.00

What is the output of the following code?

```
1: class Mammal {
2: public Mammal(int age) {
3: System.out.print("Mammal");
4: }
5: }
6: public class Platypus extends Mammal {
7: public Platypus() {
8: System.out.print("Platypus");
9: }
10: public static void main(String[] args) {
11: new Mammal(5);
12: }
13: }
```

Select one or more:

- **a. Platypus**
- **b. Mammal**
- **c. PlatypusMammal**
- **d. MammalPlatypus**
- **e. The code will not compile because of line 8.** ✓

  The code will not compile because the parent class Mammal doesn't define a no-argument constructor, so the first line of a Platypus constructor should be an explicit call to super(int age). If there was such a call, then the output would be MammalPlatypus, since the super constructor is executed before the child constructor.

- **f. The code will not compile because of line 11.**

Your answer is correct.

The correct answer is: The code will not compile because of line 8.

Question 9

Correct

Mark 1.00 out of 1.00

What modifiers are implicitly applied to all interface methods? (Choose all that apply)

Select one or more:

- [ ] **a. default**
- [ ] **b. abstract**
- [ ] **c. void**
- [ ] **d. static**
- [x] **e. public** ✓

  All interface methods are implicitly public, so option B is correct and option A is not. Interface methods may be declared as static or default but are never implicitly added, so options C and F are incorrect. Option D is incorrect—void is not a modifier; it is a return type. Option E is a tricky one, because prior to Java 8 all interface methods would be assumed to be abstract. Since Java 8 now includes default and static methods and they are never abstract, you cannot assume the abstract modifier will be implicitly applied to all methods by the compiler.

- [ ] **f. protected**

Your answer is correct.

The correct answer is: public

Question **10**

Correct

Mark 1.00 out of 1.00

Which statement(s) are correct about the following code? (Choose all that apply)

```
public class Rodent {
protected static Integer chew() throws Exception {
System.out.println("Rodent is chewing");
return 1;
}
}
public class Beaver extends Rodent {
public Number chew() throws RuntimeException {
System.out.println("Beaver is chewing on wood");
return 2;
}
}
```

Select one or more:

- [ ] **a. It will compile without issue.**
- [ ] **b. It fails to compile because the type of the exception the method throws is a subclass of the type of exception the parent method throws.**
- [x] **c. It fails to compile because the return types are not covariant.** ✓
- [ ] **d. It fails to compile because the method is protected in the parent class and public in the subclass.**
- [x] **e. It fails to compile because of a static modifier mismatch between the two methods.** ✓

The code doesn't compile, so option A is incorrect. Option B is also not correct because the rules for overriding a method allow a subclass to define a method with an exception that is a subclass of the exception in the parent method. Option C is correct because the return types are not covariant; in particular, Number is not a subclass of Integer. Option D is incorrect because the subclass defines a method that is more accessible than the method in the parent class, which is allowed. Finally, option E is correct because the method is declared as static in the parent class and not so in the child class. For nonprivate methods in the parent class, both methods must use static (hide) or neither should use static (override).

Your answer is correct.

The correct answer is: It fails to compile because the return types are not covariant., It fails to compile because of a static modifier mismatch between the two methods.